1990-09

Separation of simultaneous word sequences using Markov model techniques

Kingston, James L.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/23456

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

SEPARATION OF SIMULTANEOUS WORD SEQUENCES
USING
MARKOV MODEL TECHNIQUES

by

James L. Kingston

September, 1990

Thesis Advisor: Charles W. Therrien

Approved for public release; distribution is unlimited.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

3. DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release;
distribution is unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
Naval Postgraduate School

6b. OFFICE SYMBOL *(If applicable)*
EC

7a. NAME OF MONITORING ORGANIZATION
Naval Postgraduate School

6c. ADDRESS *(City, State, and ZIP Code)*

7b. ADDRESS *(City, State, and ZIP Code)*

8a. NAME OF FUNDING / SPONSORING ORGANIZATION

8b. OFFICE SYMBOL *(If applicable)*

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS *(City, State, and ZIP Code)*

10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
SEPARATION OF SIMULTANEOUS WORD SEQUENCES USING MARKOV MODEL TECHNIQUES

12. PERSONAL AUTHOR(S)
KINGSTON, James L.

13a. TYPE OF REPORT
Master's Thesis

13b TIME COVERED
FROM ______ TO ______

14 DATE OF REPORT *(Year, Month, Day)*
1990 September

15 PAGE COUNT
72

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the US Government.

17 COSATI CODES

FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Markov Models; text separation; Viterbi Algorithm

19 ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This thesis develops a method of separating multiple simultaneous conversations through the use of Markov Models. Text samples which represent the conversations to be used as training data are described by a grammar based upon word and word-pair occurrences within the text. This grammar is then used to establish a Markov Model for the text. These models are then combined to form a Marjov Model which describes the simultaneous occurrence of multiple conversations. Artificially generated word sequences which have the same grammar as the training conversations are supplied as input to the conversation filter, whose purpose is to "lsiten to" one of the input sequences. The conversation filter takes on either an optimal form in which the grammars of all input sequences to the filter are known, or a sub-optimal form which uses only the grammar of the desired output

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS

21 ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a NAME OF RESPONSIBLE INDIVIDUAL
THERRIEN, Charles W.

22b TELEPHONE *(Include Area Code)*
408-646-3347

22c OFFICE SYMBOL
EC/Ti

19. cont.

sequence. The conversation filter utilizes the Viterbi algorithm to extract the optimal text sequence for a best match to the grammar of the desired output. Analysis is performed to determine the efficiency of the algorithm and the performance of the algorithm for varying degrees of similarity between the grammars being separated.

Approved for public release; distribution is unlimited.

Separation of Simultaneous Word Sequences using Markov Model Techniques

by

James L. Kingston
Captain, United States Marine Corps
B.S., Rensselaer Polytechnic Institute

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1990

Author: [redacted]
James L. Kingston

Approved by: [redacted]
Charles W. Therrien, Thesis Advisor

[redacted]
Murali Tummala, Second Reader

[redacted]
Michael A. Morgan, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

This thesis develops a method of separating multiple simultaneous conversations through the use of Markov models. Text samples which represent the conversations to be used as training data are described by a grammar based upon word and word-pair occurrences within the text. This grammar is then used to establish a Markov model for the text. These models are then combined to form a Markov model which describes the simultaneous occurance of multiple conversations. Artificially generated word sequences which have the same grammar as the training conversations are supplied as input to the conversation filter, whose purpose is to "listen to" one of the input sequences. The conversation filter takes on either an optimal form in which the grammars of all input sequences to the filter are known, or a sub-optimal form which uses only the grammar of the desired output sequence. The conversation filter utilizes the Viterbi algorithm to extract the optimal text sequence for a best match to the grammar of the desired output. Analysis is performed to determine the efficiency of the algorithm and the performance of the algorithm for varying degrees of similarity between the grammars being separated.

TABLE OF CONTENTS

I. INTRODUCTION

The ability of the human ear and mind to recognize speech in a noisy environment is incredible. In a crowded room a person has the capability to focus on a single conversation and follow that conversation even if it is not the loudest speech taking place around him at the time. There are many possible clues which might enable the human mind to perform such a feat. The pitch of the speaker and the direction from which the speech arrives might be acoustic cues which aid the listener in the filtering process. Another possibility is that the listener actually hears and recognizes many words from many different conversations, and based upon an *expectation*[1] of the content of the conversation of interest selects the words which best fit that conversation. Such a method has been demonstrated to occur in the visual recognition of letters. It has been known for over 100 years that the human mind can more accurately recognize letters when seen in the context of a familiar word or pronounceable pseudo-word than when presented alone [Ref. 1]. This thesis develops a formal model for the application of such a contextual expectation to the task of separation of simultaneous conversational speech.

The ability to separate multiple simultaneous speech signals has a wide range of application. The communications industry, voice recognition and command systems, and possibly the intelligence community would all benefit. In voice communications the ability to carry many speech signals on a single channel without some form of multiplexing would result in a great cost savings. Without increasing the bandwidth of the current

[1] The word expectation is used here to indicate a state of mental receptiveness and should not be confused with the mathematical meaning of the term.

communications channels the carrying capacity of those channels would multiply directly by the number of simultaneous signals that could be separated. In voice-command systems there is currently a need for the operator to issue the commands in a noise free environment. The utility of such systems will be greatly enhanced with the ability to take them into an environment with many speakers providing background noise. The intelligence field might find it possible to determine the source of an unknown speech signal or document, or to authenticate a text from some purported source. A central assumption of this thesis is that the statistics of the speech produced by any given speaker are unique to that speaker. These statistics are called the *grammar* of the speaker. By establishing a library of these grammars it may be possible to isolate the identity of the speaker of some text by finding the optimal match between a grammar of some known source and the text. A model which is well suited to the concepts of speech separation through the use of expectations and grammars is the Hidden Markov Model.

Hidden Markov Models (HMMs) have been utilized in speech recognition systems since 1975 when James K. Baker developed the DRAGON system at Carnegie Mellon University [Ref. 2]. Since that time much successful research in speech recognition has focused on Hidden Markov Model methods. A common feature of HMM-based speech recognition systems is the use of multiple levels of decision logic or knowledge sources in recognition of a spoken word. One of these knowledge sources is grammatic syntax. The Hidden Markov Model which describes the grammar of the speech recognition system is used to choose between words which are hypothesized by other knowledge sources of the system. The use of a grammar has been shown to improve the accuracy of one continuous speech recognition system by as much as 20 to 30 percent [Ref. 3: p. 131]. In developing a model for separation of simultaneous speech the role of the grammatic syntax knowledge source is expanded by asking it to choose not between words which may be part of a single conversation, but rather between words which come from multiple conversations. Although the model developed in this thesis for speech separation does not fit the formal definition of a Hidden Markov model, it is similar enough that some methods used to analyze HMMs may be applied to the speech separation model. Further introduction to the subjects of statistical grammars, Markov models, and the selection algorithm follow.

A. STATISTICAL GRAMMARS

The grammar of a conversation is a description of the allowable combinations of words in the vocabulary which may occur within that conversation. The grammar gives a statistical description of the word sequences which make up a conversation. It is these statistics which allow the formulation of a model for syntax. In specifying a model for the syntax of a conversation there are a number of possible definitions for the grammar. Some of these are described below.

1. Word Pair Grammars

One of the simplest forms of grammar is the *word pair* grammar. In a word pair grammar all words in the vocabulary which may legally follow another word are specified. There is no weighting of the word pairs to account for frequency of use, therefore, if there are N words which might legally follow another word each of these word pairs is assigned a probability of $\frac{1}{N}$ [Ref. 3: p. 46].

2. Bigram Grammars

A slightly more complex grammar, but one which addresses the issue of frequency of use is the *bigram grammar*. A bigram grammar also specifies which words can legally follow another word in the vocabulary, but in a more quantitative probabilistic sense. That is, the probability of word W_2 following word W_1 is assigned a probability, $\Pr[W_2|W_1]$ [Ref. 3: p.47].

B. MARKOV PROCESSES

Since Markov processes and their models lie at the heart of this thesis, a basic understanding of these processes is necessary in order to comprehend the speech separation algorithm. With this in mind we present a short introduction to Markov processes.

1. The Markov Property

Markov models are representations of stochastic processes which possess the *Markov property*. A discrete parameter random process $\{X_t; t \in T\}$, where $T=\{0,1,2,...\}$, and X_t takes on values from the discrete state space $E=\{k_0,k_1,k_2,...\}$, is said to possess the Markov property if: for every $k_1,...,k_{n+1}$ in the state space E, $t_1 \le t_2 \le \cdots \le t_{n+1}$ in T, and $n=0,1,2,...$,

$$\Pr[X_{t_{n+1}}=k_{n+1}|X_{t_n}=k_n,...,X_{t_1}=k_1] = \Pr[X_{t_{n+1}}=k_{n+1}|X_{t_n}=k_n] \tag{1}$$

The Markov property (1) states that the future state of the process is conditionally independent of the past states of the process given the present state of the process [Ref. 4: p. 198]. This means that if we know the present state of a Markov process we gain no additional information about the future state of the process from knowledge of its past.

2. Discrete Markov Chains

A discrete parameter, discrete state Markov process as described above is called a *Markov Chain*. Associated with the Markov Chain is a *transition matrix* which defines the probability that the process will next be in state i given that it is currently in state j. The transition matrix P may be defined as,

$$P=(p_{ij})=\begin{vmatrix} p_{00} & p_{01} & \cdots \\ p_{10} & p_{11} & \cdots \\ \vdots & \vdots & \ddots \end{vmatrix} \qquad (2)$$

where $p_{ij}=\Pr[X_{n+1}=j \mid X_n=i]$ is called the *one-step transition probability*. Since the process must always take on some value at time $n+1$ it follows that

$$\sum_j p_{ij}=1 \qquad (3)$$

for each i in the state space. [Ref. 4: p. 201] An example of a state diagram depicting a three state Markov Chain is shown in Fig. 1.

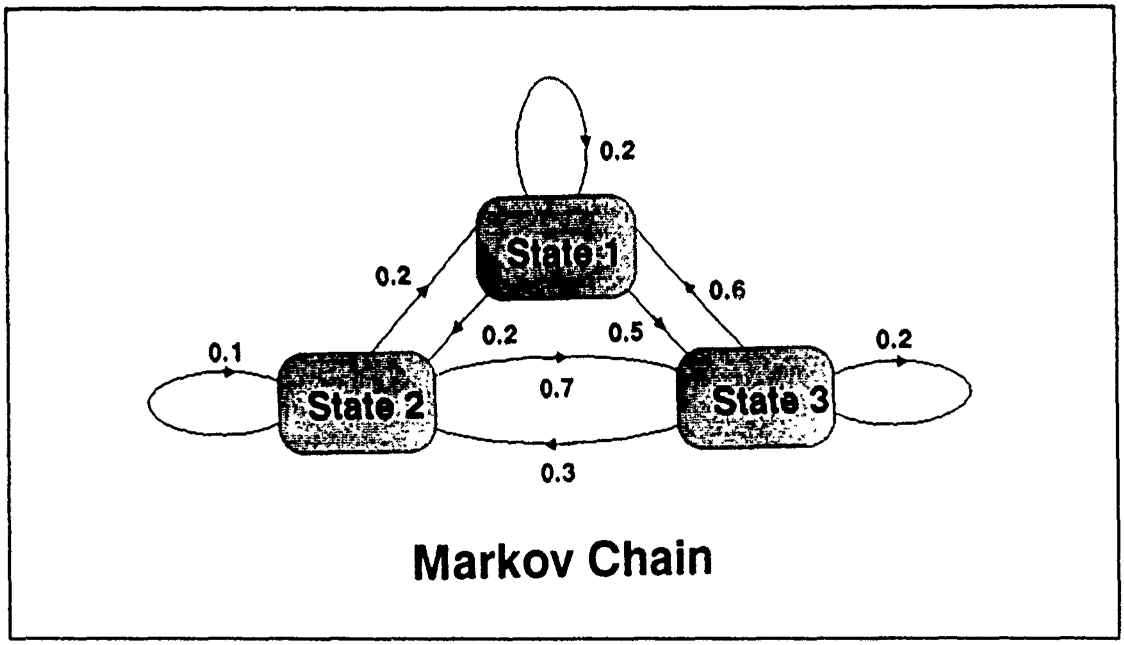

Figure 1 Markov Chain State Diagram

The diagram in Fig. 1 depicts a Markov chain with a transition matrix of

$$P=\begin{vmatrix} 0.2 & 0.2 & 0.5 \\ 0.2 & 0.1 & 0.3 \\ 0.6 & 0.7 & 0.2 \end{vmatrix}$$

which satisfies the requirement of (3).

3. Hidden Markov Models

A Hidden Markov Model is a doubly stochastic process with an underlying Markov process that is hidden, and whose effect can only be sensed through observation of another set of stochastic processes which produce the output of the system [Ref. 5: p. 5]. A schematic diagram of such a Hidden Markov Model (HMM) is given in Fig. 2.

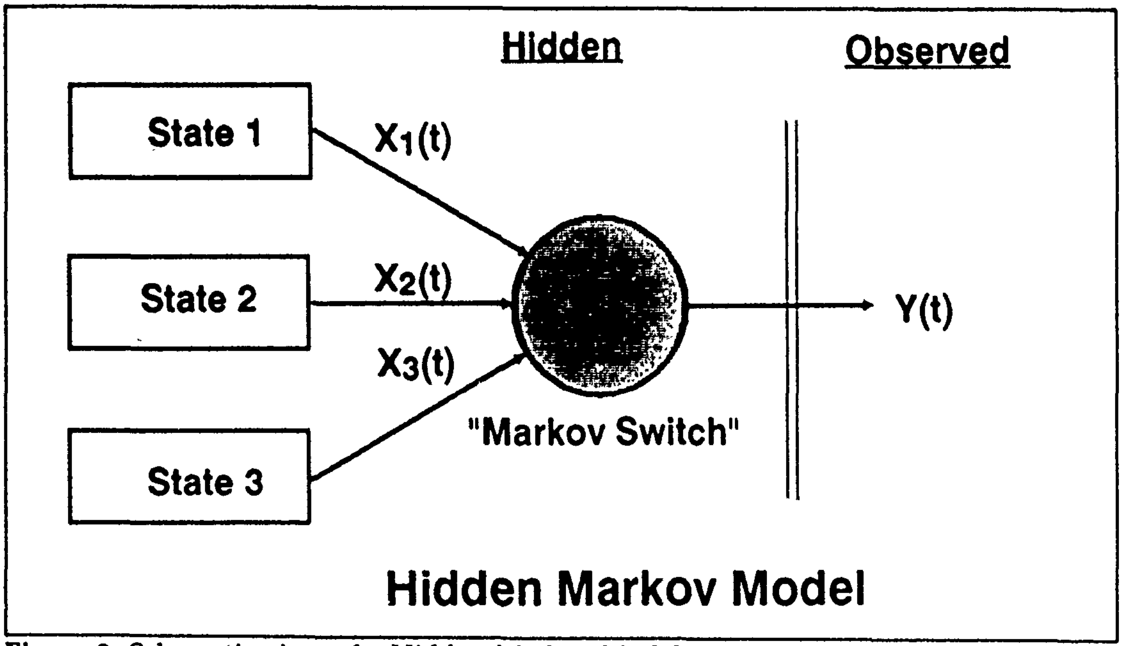

Figure 2 Schematic view of a Hidden Markov Model

Figure 2 shows a HMM with three states. Each of these states produces a random output sequence, in this case X_1(t), X_2(t), and X_3(t). The "Markov Switch" shown in Fig. 2 represents the hidden underlying stochastic process of the HMM. This switch determines which of the three random sequences produced by the states will be observed at any time t. This observed sequence is shown as Y(t) in Fig. 2. The behavior of the Markov Switch is dictated by the transition matrix describing the HMM, thus the position of the switch is actually a Markov chain.

There are four parameters required to define a hidden Markov model. These parameters are: [Ref. 5: p. 7]

- $S = \{s_1, s_2, \ldots, s_N\}$, the set of states.
- $P = \{p_{ij}\}, p_{ij} = \Pr[s_j \text{ at } t+1 \mid s_i \text{ at } t]$, the state transition probability distribution.
- $B = \{b_j(k)\}, b_j(k) = \Pr[k \text{ at } t \mid s_j \text{ at } t]$, the observation symbol probability distribution.
- $\Pi = \{\pi_i\}, \pi_i = \Pr[s_i \text{ at } t = 1]$, the initial state distribution.

A generalization of the observation symbol probability distribution permits the probability of the output symbols to be conditioned upon the occurrence of an output symbol at the previous time instant, i.e. for the output sequence to also be a Markov chain. With this generalization the observation symbol probabilities can be expressed by:

$$B = \{b_{ij}(k)\}, b_{ij}(k) = \Pr[k \text{ at } t+1 \mid k' \; at \; t, s_j \; at \; t+1, s_i \; at \; t]. \tag{4}$$

This generalization is needed for the problem we address in this thesis. From these four parameters and the observed output of the HMM, Y(t), there are three general problems that may be posed: [Ref. 3: pp. 19-25],[Ref. 5: p. 8]

- The evaluation problem - Given an observation sequence, Y(t), and the HMM parameters (P, B, Π), find the probability that the model generated the observation sequence.

- The decoding problem - Given an observation sequence, Y(t), and the HMM parameters $(\boldsymbol{P},\boldsymbol{B},\Pi)$, find the most likely (optimal) state visitation sequence that produced the observations.

- The learning problem - Given an observation sequence, Y(t), determine how the HMM parameters should be set to maximize $\Pr(\boldsymbol{Y}\,|\,(\boldsymbol{P},\boldsymbol{B},\Pi))$.

This thesis deals extensively with the second of these problems, modified to account for the Markov property exhibited by the state outputs. A well known formal technique for determining the optimal state sequence for the decoding problem exists and is known as the *Viterbi algorithm*. The next section introduces this technique.

C. THE VITERBI ALGORITHM

When presented with an observed output sequence from a Hidden Markov Model with known parameters it is often desirable to determine the optimal sequence of states visited to produce that output. If the optimality criterion chosen is one of maximum probability then the Viterbi algorithm may be used to determine that optimal state sequence. Here we present the basic algorithm followed by a description of a refinement to that algorithm.

1. Basic Viterbi Algorithm

Given some observed output sequence $\boldsymbol{Y}=\{y(1),y(2),\dots,y(N)\}$ we need to maximize the a posteriori probability:

$$\Pr[S\,|\,Y]=\frac{\Pr[Y|S]\Pr[S]}{\Pr[Y]} \tag{5}$$

where $S=\{s_1,s_2,\dots,s_N\}$ is the state visitation sequence. Since the denominator of (5) is not a function of S, maximizing $\Pr[S\,|\,Y]$ is the equivalent of maximizing the numerator. Because the distribution of y(k) is dependant upon the states of the system at time k and k-1, and the observed output at time k-1 it follows that:

$$\Pr[Y|S]=\prod_{k=1}^{N}\Pr[y(k)\,|\,y(k-1),s_{k-1},s_k] \tag{6}$$

where, for time k=1, we let

$$\Pr[y(k)\,|\,y(k-1),s_{k-1},s_k]=\Pr[y(k)\,|s_k]. \tag{7}$$

The Markov property, previously expressed in (1), provides the solution to the second half of the numerator. Following Ref. 6: pp. 190-191, since the state of the system at time k is dependant only upon the state of the system at time k-1 we have:

$$\Pr[S]=\prod_{k=1}^{N}\Pr[s_k|s_{k-1}] \tag{8}$$

where $\Pr[s_k|s_{k-1}]$ is the state transition probability p_{s_{k-1},s_k}. By combining (6) and (8) we arrive at the following equation for the maximization problem:

$$\Pr[Y|S]\Pr[S]=\prod_{k=1}^{N}\Pr[y(k)\,|\,y(k-1),s_{k-1},s_k]\Pr[s_k|s_{k-1}]. \tag{9}$$

After taking the logarithm of (9) we achieve an equivalent expression:

$$\sum_{k=1}^{N}A_{ji}(k)+B_{ji}(k) \tag{10}$$

where we have let s_k = i, and s_{k-1} = j, and have defined

$$A_{ji}(k)=\log\ (\Pr[y(k)\,|\,y(k-1),i,j]) \tag{11}$$

and

$$B_{ji}(k)=\log(p_{ji}). \tag{12}$$

It is this equation, (10), that must be maximized. [Ref. 6: p. 205]

The maximization of (10) can be thought of as finding the path through a trellis like that in Fig. 3 which has the greatest *path weight*. Path weight is defined as the sum of the weights of the branches in the path, where the values for $A_{ji}(k)$ and the values for $B_{ji}(k)$ are added to form the branch weights.[Ref. 6: p. 205]

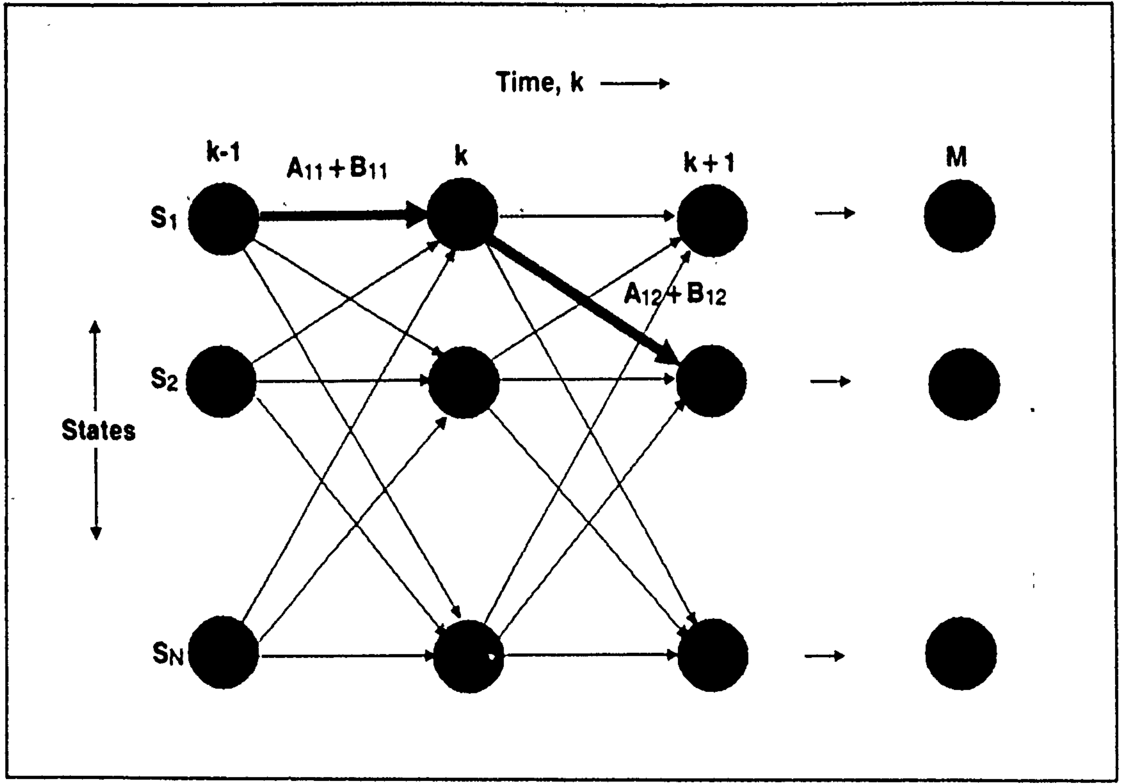

Figure 3 Trellis for the Viterbi Algorithm. After Ref. 6: p.206

It becomes apparent after considering the trellis in Fig. 3 that as the number of states in the system increases and as the number of observations in the output sequence increases the number of possible paths through the trellis becomes very large. Without some form of dynamic programming the computational expense required to enumerate the weight of each of these paths is overwhelming. The Viterbi Algorithm is just such an algorithm, and it finds the optimal path and its path weight in an almost trivial manner.

Let $\delta_i(k)$ be the weight of the optimal path leading to state i at time k, and let $\psi_i(k)$ be the state at time k-1 in the optimal path leading to state i at time k. The first step in the Viterbi algorithm is to find the initial values for the path weights at time k=1. This weight involves only the weights $A_{ji}(1)$ for each state i, as defined in (7), and the initial state distributions. From these parameters the initial path weights may be expressed as:

$$\delta_i(1) = A_{ji}(1) = \pi_i \Pr[y(k) | s_k = i]. \tag{13}$$

Since there is no previous state in the path to states at time k=1, all of the $\psi_i(1)$ are set equal to zero.

After the initialization of the algorithm for time k=1, the Viterbi Algorithm follows a recursion scheme to maintain the optimal path to each state in the trellis. The maximum weight to any given path is given by:

$$\delta_i(k) = \max_j [\delta_j(k-1) + A_{ji}(k) + B_{ji}(k)] \tag{14}$$

and the previous state in the optimal path is:

$$\psi_i(k) = \operatorname*{argmax}_j [\delta_j(k-1) + A_{ji}(k) + B_{ji}(k)] \tag{15}$$

after computing these values for each time k=2,3,...,N in the observation sequence the recursion is terminated by determining the state with the highest weight at time k=N:

$$Opt(N) = \operatorname*{argmax}_i [\delta_i(N)]. \tag{16}$$

This state is the final state in the optimal state visitation sequence. By retracing the steps taken in arriving at this final state we can reconstruct the entire optimal path. This is done by simply backtracking the previous best state variable in the following manner:

$$Opt(k)=\psi_{Opt(k+1)}(k+1) \quad \text{for } k=N-1,N-2,\ldots,1\,, \tag{17}$$

At the conclusion of this iteration the optimal state visitation sequence is contained in the variable *Opt*. [Ref. 5: p. 11]

2. Refinements to the Viterbi Algorithm.

It is possible to further reduce the amount of computation required in computing the optimal state visitation sequence by carefully monitoring the previous state variable, $\psi_i(k)$. If there is no optimal path to any state at time k which passes through state j at time k-1 then that state may be eliminated in further computations. The extreme case of this occurs when the optimal path to every state at time k passes through a single node at time k-1. In this instance all other states may be removed from further computation of the optimal path to the states at time k, and the optimal path of the system is known for all time up to k-1. [Ref. 6: pp. 206-207] This chokepoint effect is demonstrated in Fig. 4. In Fig. 4, all optimal paths from the states at time k-1 to the states at time k come from state S_2. From the description of a chokepoint given above, it follows that S_2 must be in the global optimal path. The backtracking step of the Viterbi algorithm can proceed from this known point and establish the remainder of the optimal path in the portion of the trellis that has already been processed, in this case from time t=0 to time k-1.

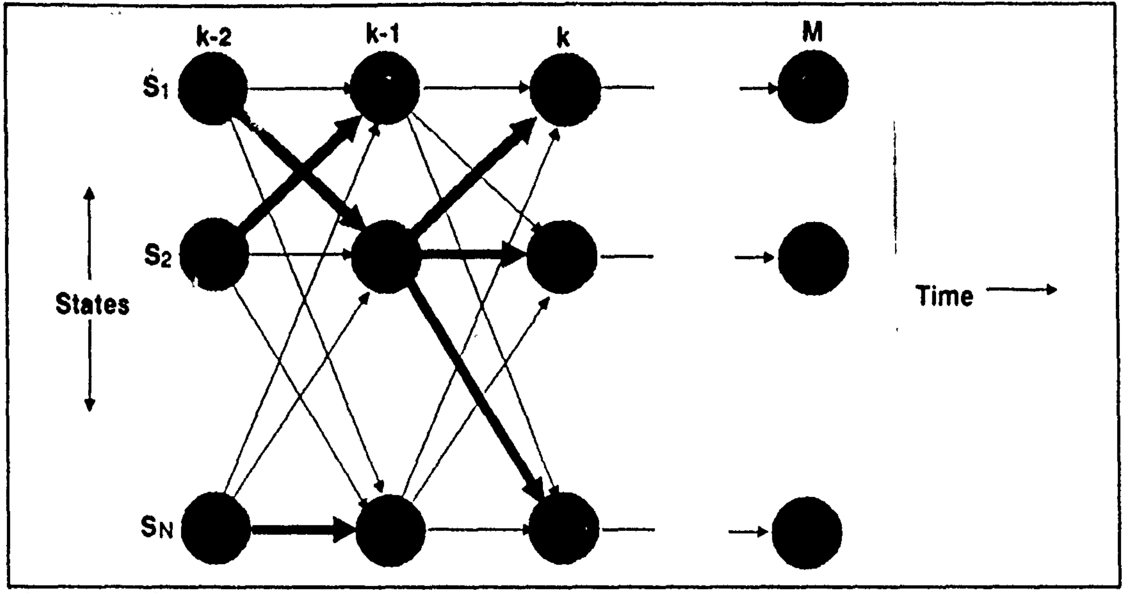

Figure 4 Chokepoint Effect in a Viterbi Trellis.

II. FILTERING SIMULTANEOUS WORD SEQUENCES

This chapter describes in detail the speech filter and its various components. As an introduction to this detailed examination a brief description of the function of the filter is provided here.

The inputs to the filter model are two or more word sequences presented at the inputs simultaneously. The filter acts upon these inputs with the objective of "listening" to one of the input sequences and discarding the others. The catch is that the input port used by each sequence may change with each time instant. Two different types of filtering are examined. The first, which is called *sub-optimal*, requires a statistical model of only the desired output sequence be available to the filter. The second type of filtering requires that statistical models for all the inputs are available to the filter. This filter type is called *optimal*. Central to the operation of both types of filter is the statistical model of the input sequence(s), called the *grammar*. The grammar or grammars are used to construct a Markov model for the filter. This in turn allows the use of the Viterbi algorithm in order to "listen" to the desired output sequence.

A. GRAMMAR

The first step in the process of filtering simultaneous speech is the establishment of a grammar which describes the statistical properties of the speech. Since the grammar selected must provide sufficient information to institute a Markov model and an associated Viterbi search for the speech it describes, bigram grammars were used for this thesis. A bigram grammar has the advantage of being simple to construct while still providing a good probabilistic description of the speech.

1. Construction of the Grammar.

Bigram grammars require that we make an estimate of the probability of word and word-pair occurrence in a sample of speech. These estimates are readily obtained by analyzing a training sample speech which has approximately the same statistical properties as the speech to be filtered.

If a word W_1 occurs N times in a training sample of length L words, then the estimated probability of occurrence for word W_1 is determined to be $\frac{N}{L}$. Since the grammar constructed will be used only to provide the parameters for the Markov model which describes the speech to be filtered it is not necessary to find an estimate of the probability of word-pair occurrence, $\Pr[W_1,W_2]$. Rather we must find an estimate for the conditional word-pair occurrence, $\Pr[W_2 \mid W_1]$. Again this is easily determined since if we know that word W_1 occurs N times and that word W_2 occurs immediately following word W_1 a total of M times then $\Pr[W_2 \mid W_1]$ is seen to be $\frac{M}{N}$.

Clearly, it would be troublesome to construct a grammar every time a filtering process is to be initiated, so the grammar must be stored on some permanent media in order to be useful. For the purposes of this thesis the grammars were stored on hard disk in the form of linked lists. The ADA source codes used to construct the grammars, PARSE.SEP and GRAMMAR.SEP, are contained on the diskette provided with this thesis. The first word in a word-pair, W_1, is stored in the word level list along with the number of times it occurs. The pair level list contains all possible following words, W_2, and the number of times those words occur after W_1. A header record is maintained to point to the first word in the word list and to account for the total number of word occurrences in the grammar. For example the grammar for the simple phrase "To be or not to be?" would be stored as shown in Fig. 5.

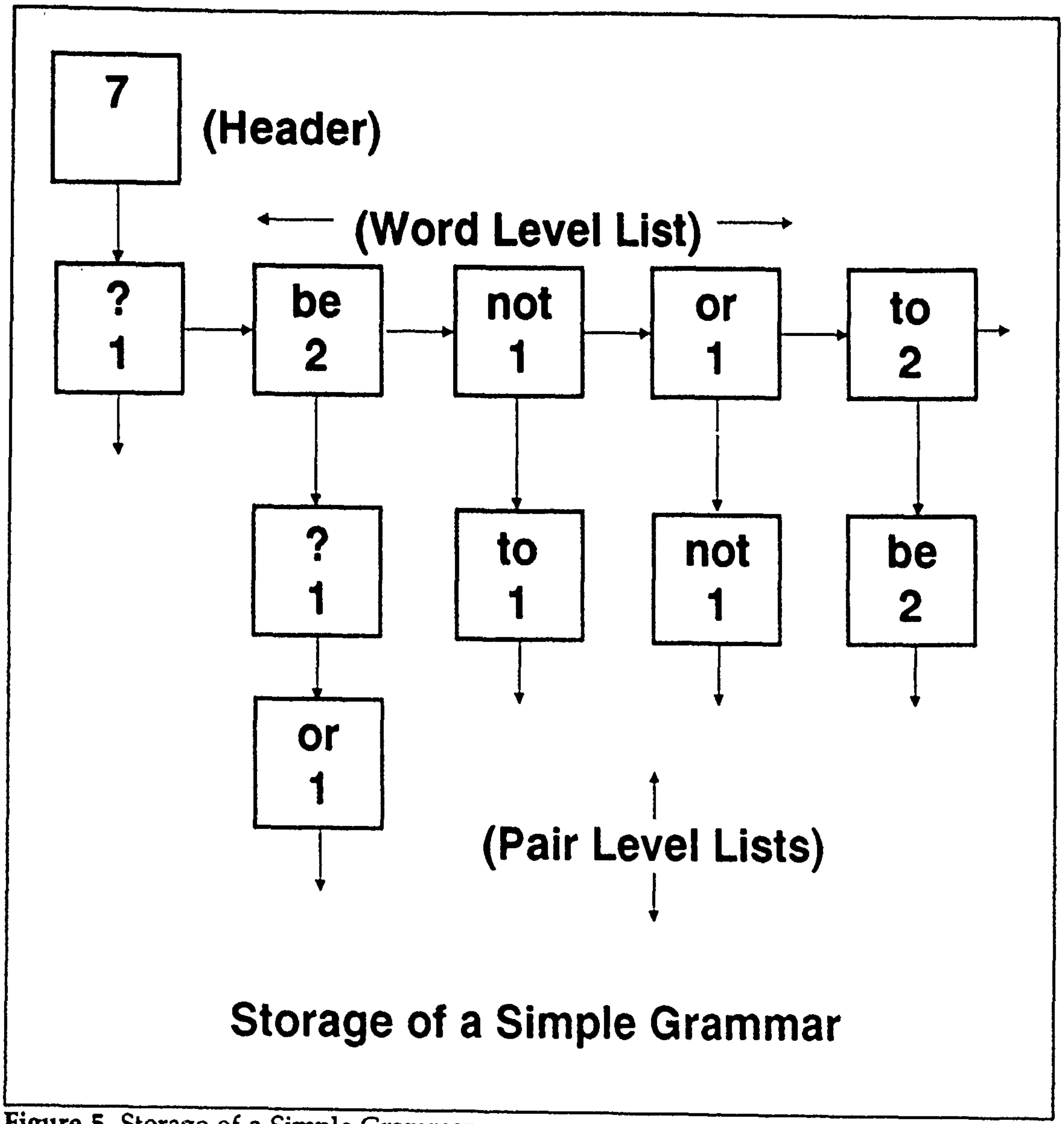

Figure 5 Storage of a Simple Grammar.

2. Parameters of the Bigram Grammars.

There are a number of parameters associated with the grammar which might have an effect on the performance of the speech filter in either speed or accuracy. In this thesis three parameters were measured, which are described below.

a. Size of the Grammar.

Grammar size is simply the total number of words in the vocabulary of the grammar. This parameter, when combined with the measure of perplexity, indicates what filter speed can be expected.

b. Measure of Perplexity.

Perplexity can be thought of as a measure of constraint placed upon a language by its grammar; that is, the number of options a grammar leaves open at any given decision point. In the case of the grammar used in this thesis, the measure of perplexity is simply a measure of, on the average, how many different words may follow the current word. This is nothing more than the number of distinct word-pairs divided by the number of distinct words in the vocabulary. [Ref. 3: p. 145-146] Perplexity is a factor in determining both speed and accuracy of the speech filter. A high perplexity value both adds to the search time required to arrive at an optimal path, and by increasing the number of possible choices in the path, serves to increase the probability of making an incorrect decision when selecting that path.

c. Similarity with Competing Grammars.

Similarity between competing grammars appears to be the single largest factor in determining the accuracy of the speech filter. Two measures of grammar similarity were established for use in this thesis.

(1) *Similarity in the Optimal Filter Grammars.* In comparing grammars in the optimal filter a measure of the strength of the intersection of the two grammars being compared must be made. Two methods of measuring the strength of the intersection were used in this thesis, the *unweighted similarity measure* , and the *weighted similarity measure.*

The unweighted similarity between two grammars is a measure of the number of distinct word-pairs the two grammars have in common. If there are 500 distinct word-pairs in the two grammars and 300 of those word-pairs are common to both grammars then the grammars are said to have an unweighted similarity of (300/500) =60.0%. This unweighted similarity provides an index of how often word-pairs common to both grammars may occur in speech appearing at the input of the filter process. One disadvantage of the unweighted similarity measure is that not all the information available to us is taken into account. Since a bigram grammar is being used as the basis of the speech model, there is probabilistic data available which might serve to enhance the similarity measure.

Weighted Similarity is measured in much the same fashion as unweighted similarity except that frequency of occurrence for each word and word-pair is taken into account, and the total number of occurrences, rather than distinct occurrences, is used. If, for example, the word-pair "the man" occurs 50 times in grammar 1 and 30 times in grammar 2, an unweighted similarity measure counts only a single common word-pair. In computing a weighted similarity measure this same word-pair accounts for 80 common word-pairs between the two grammars. Therefore, more frequently occurring word-pairs contribute more to the similarity measure than do less frequently occurring ones, and a more realistic measure of similarity results.

(2) Similarity in Sub-Optimal Filter Grammars. For sub-optimal filter grammars the degree of similarity is a measure of how close the competing grammar is to being a subset of the desired grammar. An example of how the unweighted sub-optimal similarity is computed follows. If the desired grammar has 350 distinct word-pairs, the competing grammar has 450 distinct word-pairs, there are 300 word-pairs common to the two, and a total of 500 distinct word-pairs in both grammars, the competing grammar has an unweighted word-level similarity of $300/450=66.6\%$. The weighted similarity measures for the sub-optimal case are computed in the same manner except that word-pair frequency of occurrence is taken into consideration.

3. Using a Grammar.

There are two ways in which a grammar is used in this thesis. The first is to generate artificial speech sequences which fit the same statistical pattern as the training speech; the second is to determine the word and word-pair probabilities for speech sequences presented at the input of the speech filter. Methodology of use and computational expense incurred in those uses are discussed. The reader may wish to refer back to Fig. 5 in order to follow the steps described below.

a. Generation of Artificial Speech.

The purpose of generating artificial speech is to make available speech that is different than that used as training data but still possesses the same statistical qualities. By selecting words and word-pairs from a grammar under the constraints imposed by that grammar, artificial speech which has the desired statistical characteristics can be created.

The header record for each grammar contains the total number of word occurrences, N, in that grammar. If a random number generator with a uniform distribution is used to select a number M, where $1 \leq M \leq N$, and the word level linked list is traversed until the total number of word occurrences seen is greater than or equal to M, and the word at that list node is selected, then the words selected will approximate the statistical distribution of words in the training data. Selection of such a word from an ordered linked list will take an average of $\frac{V}{2}$ look-ups[2], where V is the number of words in the vocabulary of the grammar in question.

Once the first word in the artificial speech sequence is established it is necessary to use the conditional probabilities for word-pairs in order to generate all further words in the sequence. To find the next word in the sequence the initial word is used as the first word of a word-pair and this first word is located in the word level linked list. This again takes an average of $\frac{V}{2}$ look-ups. The word level node contains a field which indicates how many occurrences of that word were in the training data. If there were K occurrences of the initial word in the training data, the uniformly distributed random number generator is used to produce a number M, where $1 \leq M \leq K$. The pair level list is then traversed until the total number of pairs seen is greater than or equal to M. The word at that node is determined to be the second word in the word-pair. This process continues with the newly determined word becoming the first word in the word-pair until a sequence of the desired length has been generated. Notice that the next word in the sequence depends only upon the previous word. This indicates that the generated word sequences are in fact Markov chains, with each word representing a state of the chain.

The measure of perplexity, Q, is the average number of nodes in the pair level list, and an average of $\frac{Q}{2}$ look-ups must be made to find the randomly selected

[2] Look-up is used here to describe the act of accessing the information contained in a single record of the linked list. This may require access to disk, RAM, or some other storage media.

second word of the word pair. Thus the total average number of look-ups required to find the second and successive words in the artificial speech sequence is $\frac{V+Q}{2}$.

b. Determination of Word and Word-Pair Probabilities.

When attempting to separate the word sequences presented at the input of the speech filter it is necessary to determine the word probability of the presented words and the word-pair probabilities of all possible pairs at the input. Since this accounts for the majority of the processing required to perform the filtering operation, it is quite important to have an understanding of how these probabilities are obtained.

To determine word probability the grammar header node is accessed to determine the total number of word occurrences, N, in the training data. The word level list is traversed until the word of interest is located and the number of occurrences of that word, K, is read. The estimated word probability is then $\frac{K}{N}$. As in finding the first word in an artificially generated speech sequence, the average number of look-ups required to find a word probability for a word which exists within the vocabulary of the grammar is $\frac{V}{2}$. Because an ordered list is used to maintain the grammar on disk, even words which are not members of the vocabulary can be dealt with in this same average number of look-ups. When a word does not occur in the vocabulary a probability of zero is returned.

Word-pair probabilities are found by first traversing the word level list until the first word in the word-pair is located. Associated with this word is its number of occurrences, K. The pair level list is then traversed until the second word in the pair is located and its number of occurrences, L, is found. The estimated conditional word-pair probability is $\frac{L}{K}$. The average number of look-ups required to find a word-pair probability is $\frac{V+Q}{2}$, where Q is the perplexity of the grammar. In the event that either the first or second word of a word-pair is not found in the grammar a probability of zero is returned.

Of great importance in the implementation of the filter is the fact that these word-pair probabilities represent the probability of transitioning between the states of the Markov chains symbolized by the artificially generated word sequences.

B. FORMULATION OF THE FILTER MODELS

1. Definition of the Filter Problem.

Figure 6 presents a diagram of the filtering problem encountered in the separation of simultaneous word sequences.

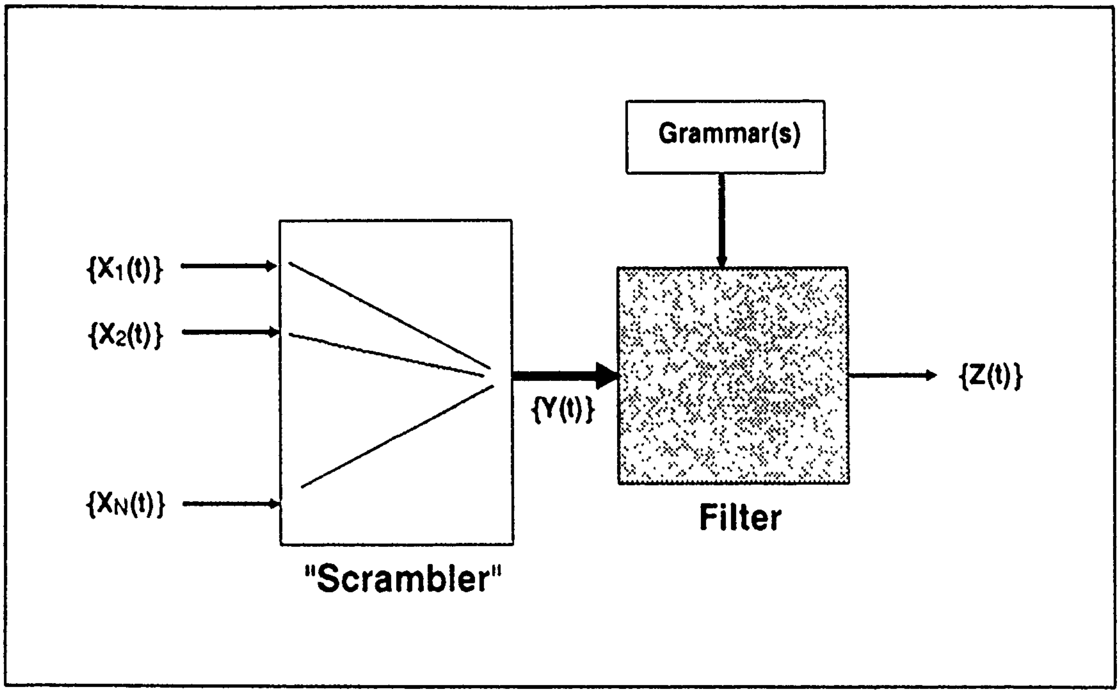

Figure 6 Block Diagram of the Filtering Process.

Multiple, independent, random sequences, $\{x_1(t)\},\{x_2(t)\},\dots,\{x_N(t)\}$, all of which exhibit the Markov property (1), are input to a "scrambler" which is "white" in nature. One of these inputs is the desired sequence $\{x_d(t)\}$. The output of this scrambler, $\{\bar{y}(t)\}$, is a random rearrangement of the elements of the inputs (i.e. words) at a single time instant.

For example if two input sequences, $x_1(k)$ and $x_2(k)$, are presented to the scrambler at time k, the output of the scrambler at time k will be either:

$$\vec{y}(k)=\begin{bmatrix}x_1(k)\\x_2(k)\end{bmatrix}\ or\ \ \vec{y}(k)=\begin{bmatrix}x_2(k)\\x_1(k)\end{bmatrix}$$

and these two possibilities occur with equal probability. These rearrangements of the input sequence order will be defined as the states of the system. Further definition of these states follows later in this thesis, but for the purpose of deriving the optimization problem for this system this definition will suffice.

The output of the scrambler, $\{\vec{y}(t)\}$, is the input to the speech filter which attempts to reconstruct the desired sequence. The reconstruction of $\{x_d(t)\}$ is to be optimal in terms of maximizing the probability that the output of the speech filter, $\{z(t)\}$, is equal to $\{x_d(t)\}$. The system described here fits the generalized model given in section I.B.3 and the optimal state sequence of this system may be obtained through use of the Viterbi algorithm as described in section I.C.1. Because the scrambler does not possess the Markov property and the states output from the scrambler are uniformly distributed the Viterbi algorithm can be further simplified as shown below.

The optimization problem for the hidden Markov model was given in equations (5) through (12) in section I.C.1. While these equations characterize a system with an underlying state sequence that is in general Markov, a special case arises when the states of the system are independent. The model described above exhibits such an independence of system states, and in this special case the probability of a state sequence is:

$$\Pr[S]=\prod_{k=1}^{N}\Pr[s_k] \tag{18}$$

By substituting (18) into (9) the equation to be maximized is obtained as follows:

$$\Pr[Y|S]\Pr[S]=\prod_{k=1}^{N}\Pr[\bar{y}(k)\,|\,\bar{y}(k-1),s_k,s_{k-1}]\Pr[s_k]. \tag{19}$$

Because the state distribution in this problem is uniform, every state sequence has equal probability and Pr[*S*] may be ignored in the maximization. With this observation we arrive at the final path weight equation:

$$\Pr[Y|S]=\prod_{k=1}^{N}\Pr[\bar{y}(k)\,|\bar{y}(k-1),s_k,s_k-1]. \tag{20}$$

After taking the logarithm of (20) as was done to arrive at (10) we have the equivalent expression:

$$\sum_{k=1}^{N}A_{ji}(k) \tag{21}$$

where we have let s_k = i, and $s_{k\text{-}1}$ = j, and have defined

$$A_{ji}(k)=\log(\Pr[\bar{y}(k)\,|\bar{y}(k-1),j,i]). \tag{22}$$

The probabilities required to implement the Viterbi algorithm using the path weight equation (21) are available through analysis of the observed output of the scrambler using the grammar(s) associated with the various input sequences. The way in which this analysis is performed determines whether the filter is sub-optimal or optimal in nature. These two different filters are described fully in the following sections.

2. The Sub-Optimal Solution.

The first filter to be discussed is the sub-optimal filter. This filter model is based upon *a priori* knowledge of the grammar of the desired output sequence only.

a. The Model for the Sub-Optimal Filter.

The key to determining the solution to the filter problem lies in the definition of the states of the model. In this problem all the word sequence elements input to the scrambler can be observed at each time instant; however, which element belongs to which input sequence is unknown. By defining the observed output states as the permutations of the inputs, as we have previously indicated, a model which fits the problem definition may be developed. In the sub-optimal filter there is only one grammar and only one input sequence which matches that grammar. The problem is, given the set of observed words $y_1(t),y_2(t),...,y_N(t)$ at time t, which of these matches the grammar in the optimal way. Since there are N input sequences there are N possible positions that the desired word might take in the output vector. Because we don't really care what positions in the output vector all the undesired words take, the N possible positions that the desired word might occupy define the N states of the model. The resultant model for an N input sub-optimal filter using this state definition is depicted in Fig. 7.

The notation used to define the states of the sub-optimal filter associates one of the output vector positions with the desired grammar, leaving the other positions undefined. For example state two of an N state model has the notation $[x_1,(x_2,g),x_3,...,x_N]$ where the input paired with the grammar, g, is the desired input. The notation given for state two represents an observed output vector:

$$\vec{y}(k)=\begin{vmatrix} x_?(k) \\ x_d(k) \\ x_?(k) \\ \vdots \\ x_?(k) \end{vmatrix}$$

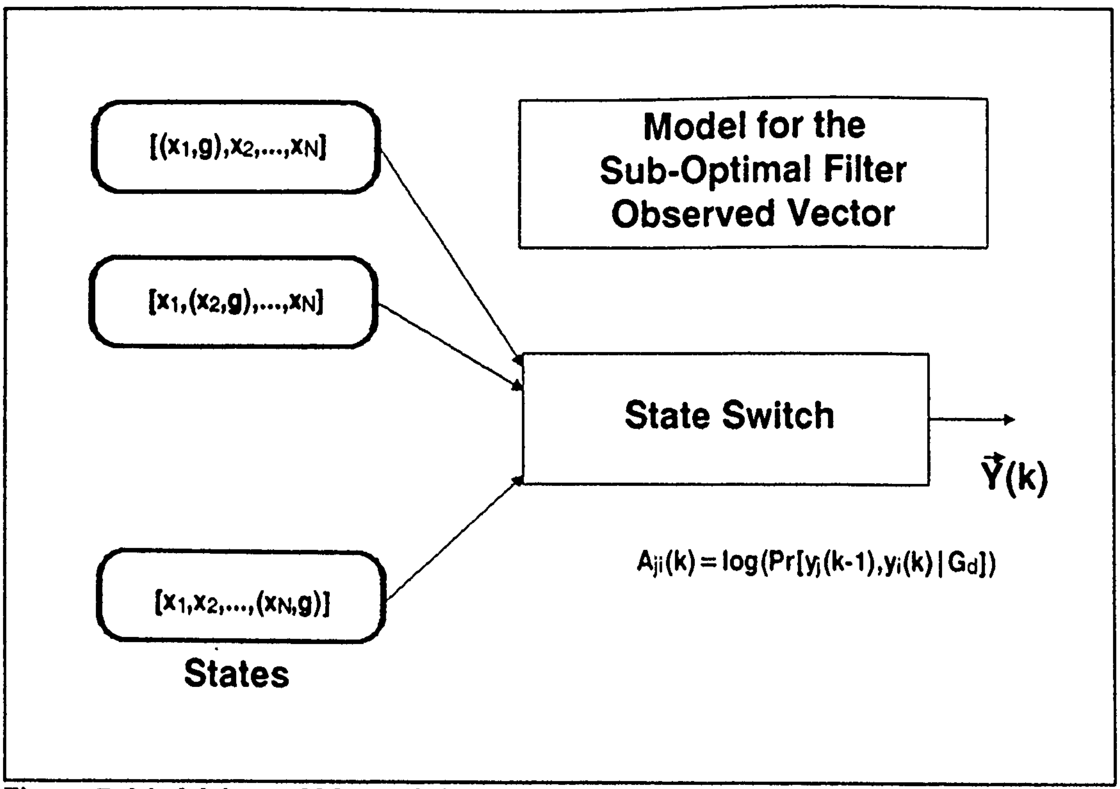
Model for the Sub-Optimal Filter Observed Vector
$[(x_1,g),x_2,\ldots,x_N]$
$[x_1,(x_2,g),\ldots,x_N]$
$[x_1,x_2,\ldots,(x_N,g)]$
States
State Switch
$\vec{Y}(k)$
$A_{ji}(k)=\log(\Pr[y_j(k-1),y_i(k)\,|\,G_d])$

Figure 7 Model for an N-Input Sub-Optimal Filter.

Once the states have been defined, in order to implement the Viterbi algorithm it is necessary to establish the values of $A_{ji}(k)$ for all k, j, and i. The observed word vectors, $\vec{y}(t)$, at the output of the scrambler, when analyzed under the constraints of the desired word sequence grammar, provide the information necessary to determine these values.

If we let $\alpha(y_j(k-1),y_i(k))$ be the logarithm of the word-pair probability of the word-pair formed by $y_j(k-1)$ and $y_i(k)$ under the constraints of the desired sequence grammar, then for each i and j where i,j = 1,2,...,N :

$$A_{ji}(k)=\alpha(y_j(k-1),y_i(k))=\log\{\Pr[y_j(k-1),y_i(k)\,|\,G_d]\} \qquad \textbf{(23)}$$

where G_d is the desired grammar. Notice that for an N-input filter there are N^2 word-pair probabilities that need to be computed. At time k = 1 there is no word-pair formed so we let $A_{ji}(1)$ be defined as the logarithm of the single word probability of $y_i(1)$, for i = 1,2,...,N. These concepts are further explained in the following example of a two input sub-optimal filter.

Let $x_1(k-1), x_1(k)$ be two words from sequence x_1, and $x_2(k-1), x_2(k)$ be two words from sequence x_2. If the output of the scrambler is:

$$\bar{y}(k-1) = \begin{vmatrix} x_2(k-1) \\ x_1(k-1) \end{vmatrix}, \quad \bar{y}(k) = \begin{vmatrix} x_1(k) \\ x_2(k) \end{vmatrix}$$

then the four word-pair probabilities under the desired grammar are:

$$A_{11}(k) = \alpha[x_2(k-1), x_1(k)];$$

$$A_{12}(k) = \alpha[x_2(k-1), x_2(k)];$$

$$A_{21}(k) = \alpha[x_1(k-1), x_1(k)];$$

$$A_{22}(k) = \alpha[x_1(k-1), x_2(k)].$$

If x_1 is the desired sequence, then we would expect that $A_{21}(k)$ would give the highest probability. From the depiction in Fig. 8 of the two input filter example above, it can be seen that these conditional word-pair probabilities correspond closely to the state transition probabilities of a HMM, and play the same role in the Viterbi trellis. The implementation of the Viterbi algorithm to determine the optimal word sequence is detailed below.

b. *The Viterbi Algorithm for the Sub-Optimal Case.*

To implement the Viterbi algorithm for the sub-optimal filter solution, substitution of the path weight equation for the special case model, (21), must be made for the generalized path weight equation, (10), in the algorithm presented in section I.C.1. In the special case path weight equation there is no term, $B_{ji}(k)$, which corresponds to the state transition probabilities. This is because p_{ji} in (12) becomes $\Pr[s_i]$, which, because the

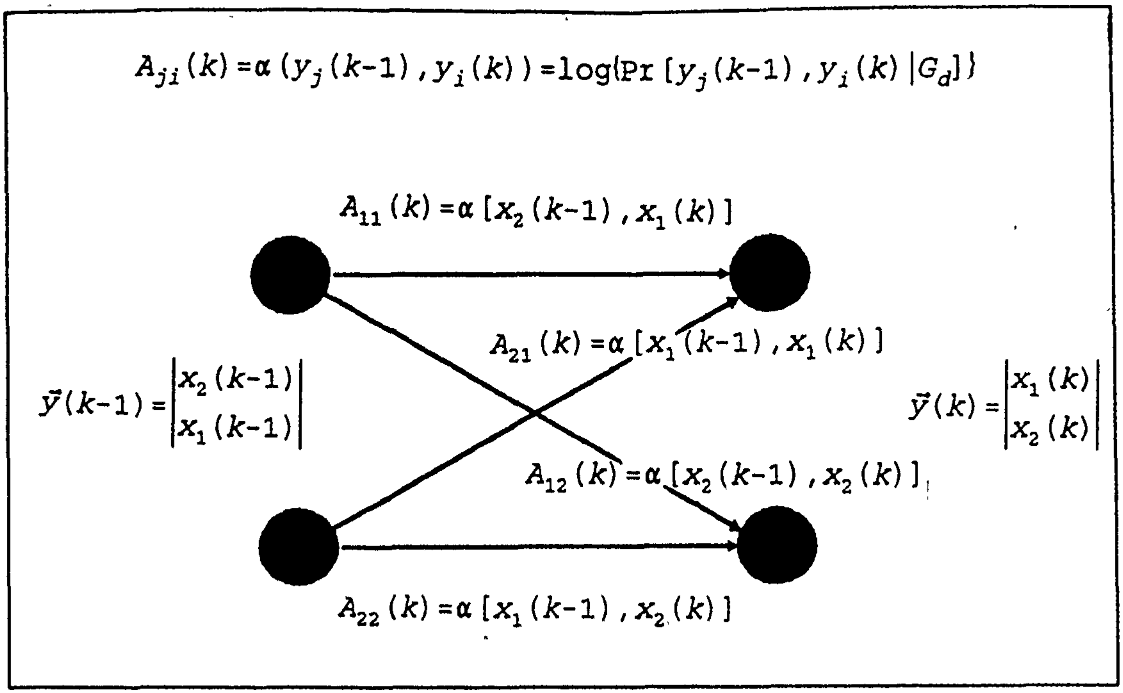
$A_{ji}(k) = \alpha(y_j(k-1), y_i(k)) = \log\{\Pr[y_j(k-1), y_i(k) | G_d]\}$
$A_{11}(k) = \alpha[x_2(k-1), x_1(k)]$
$A_{21}(k) = \alpha[x_1(k-1), x_1(k)]$
$\vec{y}(k-1) = \begin{vmatrix} x_2(k-1) \\ x_1(k-1) \end{vmatrix}$
$\vec{y}(k) = \begin{vmatrix} x_1(k) \\ x_2(k) \end{vmatrix}$
$A_{12}(k) = \alpha[x_2(k-1), x_2(k)]$
$A_{22}(k) = \alpha[x_1(k-1), x_2(k)]$

Figure 8 State Transitions for a Two Input Sub-Optimal Filter.

distribution of the scrambler is uniform, is a constant for all i. Since this value is the same for all i the state probability distribution does not affect the optimization and may be ignored. The Markov properties of the state output sequences, as defined in (6) and (22), for the sub-optimal filter are provided by the conditional word-pair probabilities in equation (23). With these definitions, the path weight equation to be maximized for the sub-optimal filter becomes:

$$\sum_{k=1}^{M} \alpha(y_j(k-1), y_i(k)) \tag{24}$$

and the Viterbi trellis node and branch weights are shown in Fig. 9. Notice that a dummy node must be introduced at time t=0 in order to provide branch weights to the states at time t=1.

The Viterbi algorithm for the sub-optimal filter case is obtained by using the Viterbi trellis of Fig. 9 and substituting the branch weights as required into equations

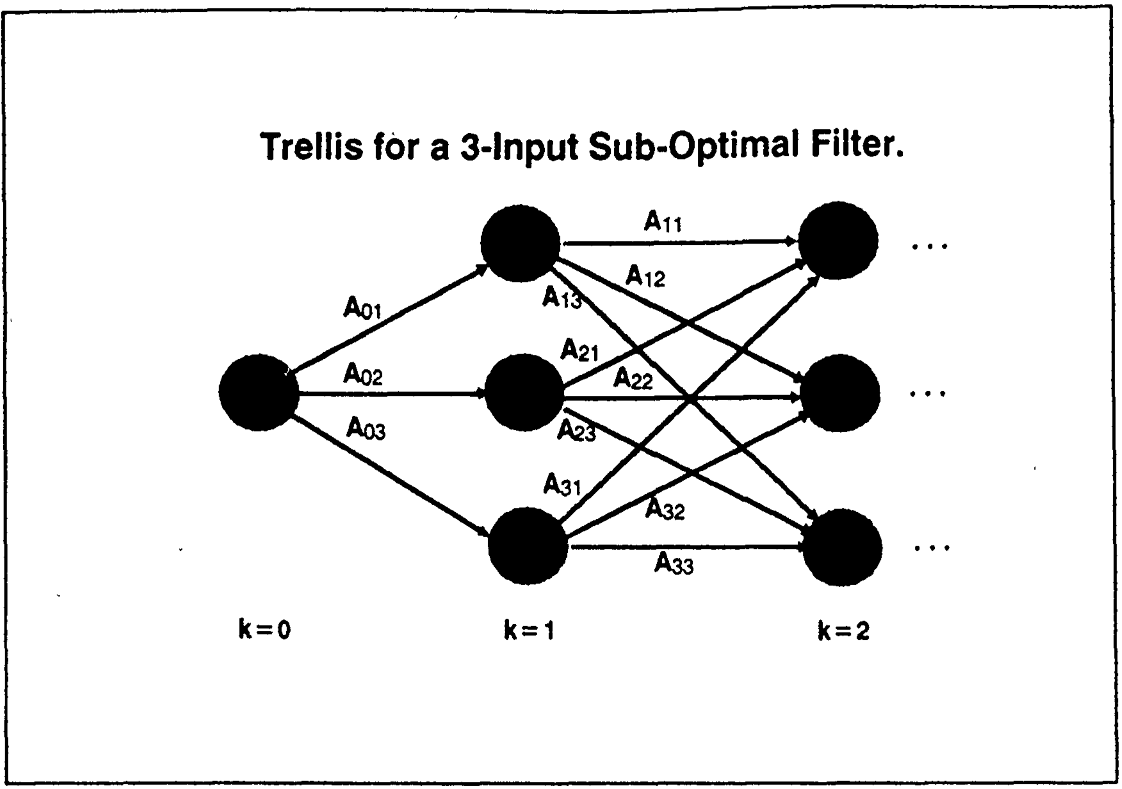

Figure 9 Viterbi Trellis Weights for the Sub-Optimal Filter.

(13) - (17). The initialization equation (13), must account for the branch weights now in place for the transition from time instant zero to time instant one. With this substitution the initialization equation is:

$$\delta_i(1)=A_{0i}(1)=\Pr[y_i(1)\,|\,G_d] \tag{25}$$

The initial values of $\psi_i(1)$ must still be set to zero for all i. The recursive portions of the algorithm, equations (14) and (15), after substitution of (23) are:

$$\delta_i(k)=\max_j\,[\delta_j(k-1)+\alpha(y_j(k-1),y_i(k))] \tag{26}$$

$$\psi_i(k)=\operatorname*{argmax}_j[\delta_j(k-1)+\alpha(y_j(k-1),y_i(k))] \tag{27}$$

where k = 2,3,...,M and i,j = 1,2,...,N. Incorporation of the "chokepoint" refinement described in section I.C.2 is accomplished by testing $\psi_i(k)$ after each iteration of the recursion. If, for all i and j, $\psi_i(k)=\psi_j(k)$ then a chokepoint has been reached and backtracking to determine the optimal state visitation sequence up to time k-1 may occur.

The chokepoint refinement to the Viterbi algorithm requires some additional bookkeeping in the backtracking step. There are two occasions on which backtracking may be initiated, namely at the end of the input sequence, or when a chokepoint has been detected. An additional variable representing the last known point in the optimal state sequence must be maintained. Let this variable be called R_l and be set to zero upon initialization of the algorithm. If backtracking is initiated because the end of the word sequence has been reached then no modification of (16) is required and (17) may be executed immediately. However, if backtracking is initiated because a chokepoint has been discovered then the following actions must occur:

- Set the optimal path at time k-1 to $Opt(k-1)=\psi_1(k)$, the chokepoint node.
- Set the lower limit on the backtracking to the previous entry time to the recursion. This time is R_l+1.

Once this bookkeeping has been completed the backtracking equation, (17), can be executed with a modification of the limits on t. With this modification (17) becomes:

$$\text{for } t=k-1,k-2,\ldots,R_l+1 \quad Opt(t)=\psi_{Opt(t+1)}(t+1) \tag{28}$$

The next step is dependant upon the reason for the initiation of backtracking. If backtracking was initiated because of a chokepoint condition then the recursion step is re-entered with the lower limit on k in equations (26) and (27) reset to k+1 and the lower bound on the backtrack step reset so that R_l=k-1. If backtracking was initiated because of an end of input sequence condition then the algorithm is complete and the optimal word sequence, $\{z(t)\}$, has been determined.

c. *Calculation Requirements.*

The purpose of this section is to examine the number of calculations required to produce the optimal word sequence using the algorithm described above. The calculation requirements will be measured in terms of look-ups and floating point operations, such as multiplications and logarithms.

Assume the problem has the following characteristics:

- N input sequences.
- Input sequences have length M.
- Desired grammar has a vocabulary of V distinct words.
- Desired grammar has a perplexity of Q.

Initialization of the Viterbi algorithm requires that N single word probabilities be established. To establish a single word probability requires an average of $\frac{V}{2}$ look-ups and one floating point operation (flop). The logarithm of the word probability must be obtained adding another expense to each word probability look-up. The resulting average number of operations required for initialization is $N\frac{V}{2}$ look-ups, N flops, and N logarithms. The recursion equations utilize N^2 word-pair probabilities. Each word-pair probability requires an average of $\frac{V+Q}{2}$ look-ups, one flop, and one logarithm. The recursion is carried out M-1 times for a total of $(M-1)N^2\frac{V+Q}{2}$ look-ups, $(M-1)N^2$ flops, and $(M-1)N^2$ logarithms. Since all values for $\psi_t(k)$ are stored in memory as they are calculated there are no floating point operations or look-ups required to perform the backtracking step.

Combining all the calculations required in the steps described above produces the total calculation requirements for the sub-optimal filter:

$$\text{look-ups} = (M-1)\frac{V+Q}{2}N^2 + \frac{V}{2}N \tag{29}$$

$$\text{flops} = (M-1)N^2 + N \tag{30}$$

$$\text{logarithms} = (M-1)N^2 + N \tag{31}$$

These figures are quite important when comparing the performance of the sub-optimal filter with that of the optimal filter.

3. The Optimal Solution.

The second filter type to be discussed is called the optimal filter. This filter model is based upon *a priori* knowledge of the grammars of each input sequence presented to the filter.

The method used to define the states of the optimal filter is similar to that used for the sub-optimal filter. Each state is a possible permutation of the input sequences, except in this case the position of each grammar and each input element must be considered.

a. The Model for the Optimal Filter.

In the optimal filter there is a matching grammar for each input sequence. This filter model should provide a lower probability of error since the information contained in each grammar is working toward optimizing the solution. Instead of determining the position in the output vector which matches just the desired input word, the problem is now to determine which output element positions match each input grammar in the optimal way. If there are N input words then there are N possible positions that the desired word could take in the output vector. This leaves N-1 positions for the second input word, N-2 for the third and so on until the Nth input word has only a single position. From this analysis it follows that for N input sequences there are N! different output possibilities and therefore N! states in the optimal filter model. This differs from the sub-optimal case where only the position of the desired grammar is important. For example, if $\{x_1(t)\}$ is the desired sequence, the output vectors

$$\bar{y}(t)=\begin{vmatrix}x_1(t)\\x_2(t)\\x_3(t)\end{vmatrix} \text{ and } \bar{y}(t)=\begin{vmatrix}x_1(t)\\x_3(t)\\x_2(t)\end{vmatrix}$$

would be considered a single state in the sub-optimal filter, but the optimal filter differentiates between the two due to the change in the positions of the competing grammars. Throughout this section a three input optimal filter will be utilized as an example and the model for such a filter is depicted in Fig. 10. The notation used to define the states of the optimal filter associates each of the output vector positions with one of the input grammars. For example state two of the three input filter model has the notation $[(x_1,G_1),(x_2,G_3),(x_3,G_2)]$, where G_1, G_2, and G_3 are the three filter grammars. The state given by the notation of this example represents the observed output vector:

$$\bar{y}(k)=\begin{vmatrix}x_1(k)\\x_3(k)\\x_2(k)\end{vmatrix}.$$

Once the states have been defined, it is necessary to establish the values of $A_{ji}(k)$ for ali k, j, and i, in order to implement the Viterbi algorithm. As in the sub-optimal case, the observed word vectors, $\bar{y}(t)$, at the output of the scrambler provide the information necessary to determine these values. In the optimal filter the constraints applied to the analysis of the observed word vectors are comprised of a combination of the grammars of all the input sequences, not just the desired sequence. The probability that

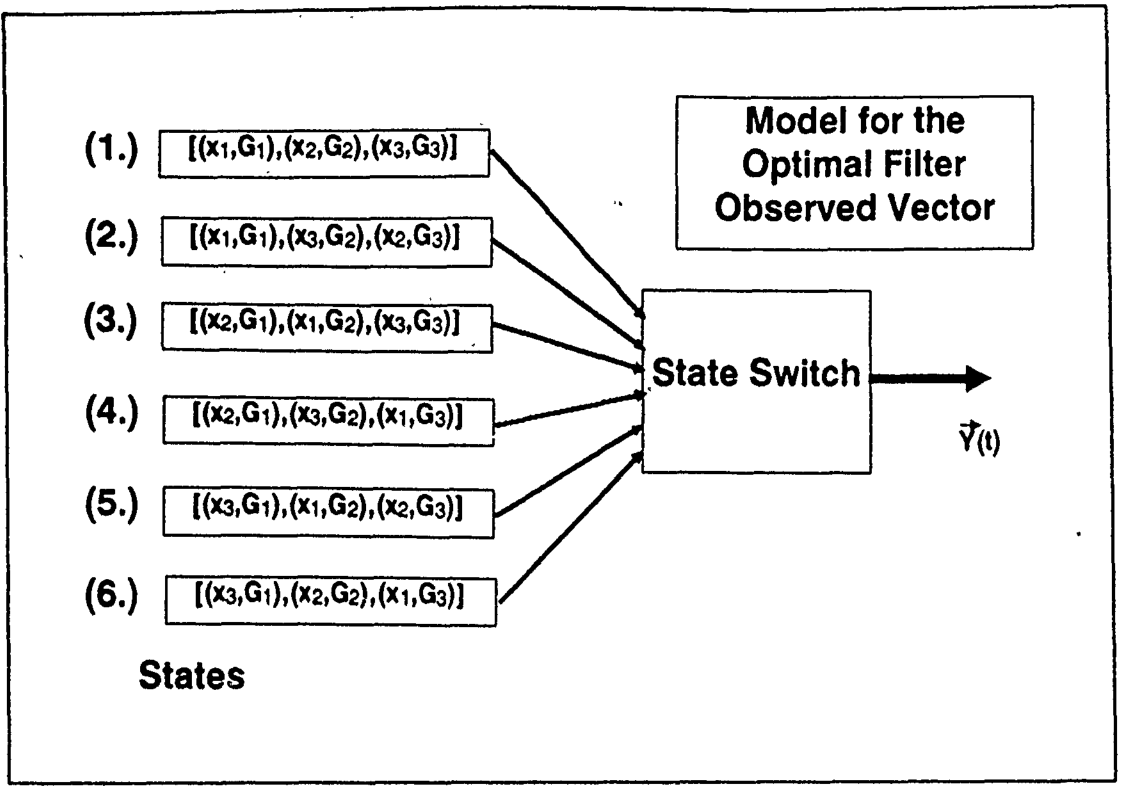
Model for the Optimal Filter Observed Vector
(1.) [(x1,G1),(x2,G2),(x3,G3)]
(2.) [(x1,G1),(x3,G2),(x2,G3)]
(3.) [(x2,G1),(x1,G2),(x3,G3)]
(4.) [(x2,G1),(x3,G2),(x1,G3)]
(5.) [(x3,G1),(x1,G2),(x2,G3)]
(6.) [(x3,G1),(x2,G2),(x1,G3)]
States
State Switch
Y(t)

Figure 10 Model for a 3 Input Optimal Filter.

$\vec{y}(k-1)$ is from state i and $\vec{y}(k)$ is from state j is a joint probability of the word-pairs associated with that particular state transition. For example, given the three-input filter of Fig. 10, the transition from state two to state three involves the three word-pairs $(y_1(k-1),y_2(k))$, $(y_2(k-1),y_3(k))$, and $(y_3(k-1),y_1(k))$, where $(y_1(k-1),y_2(k))$ is constrained by grammar G_1, $(y_2(k-1),y_3(k))$ is constrained by grammar G_3, and $(y_3(k-1),y_1(k))$ is constrained by grammar G_2. Since the input word sequences are independent, the joint probability can be obtained by forming the product of the individual word-pair probabilities. If $\alpha_{G_n}(y_q(k-1),y_r(k))$ is the logarithm of the word-pair probability of the word-pair formed by $y_q(k-1)$ and $y_r(k)$ under the constraints of the grammar associated with input n, for each q, r, and n where q,r,n = 1,2,...,N, then the joint probability required to determine the output state transition probability in the example above is:

$$A_{23}(k)=\alpha_{G_1}(y_1(k-1),y_2(k))+\alpha_{G_2}(y_3(k-1),y_1(k))+\alpha_{G_3}(y_2(k-1),y_3(k)).$$

With an N input filter there are N! states, and $(N!)^2$ output state transition probabilities. Even the three-input filter of Fig. 10 with 36 possible state transitions is too complex to completely enumerate here. A portion of this set of transitions is listed here in order to give a general idea of how these state transitions are calculated. Referring to Fig. 10 for the state definitions, the transition probabilities from state 2 to all other states are:

$$A_{21}(k)=\alpha_{G_1}(y_1(k-1),y_1(k))+\alpha_{G_2}(y_3(k-1),y_2(k))+\alpha_{G_3}(y_2(k-1),y_3(k))$$

$$A_{22}(k)=\alpha_{G_1}(y_1(k-1),y_1(k))+\alpha_{G_2}(y_3(k-1),y_3(k))+\alpha_{G_3}(y_2(k-1),y_2(k))$$

$$A_{23}(k)=\alpha_{G_1}(y_1(k-1),y_2(k))+\alpha_{G_2}(y_3(k-1),y_1(k))+\alpha_{G_3}(y_2(k-1),y_3(k))$$

$$A_{24}(k)=\alpha_{G_1}(y_1(k-1),y_2(k))+\alpha_{G_2}(y_3(k-1),y_3(k))+\alpha_{G_3}(y_2(k-1),y_1(k))$$

$$A_{25}(k)=\alpha_{G_1}(y_1(k-1),y_3(k))+\alpha_{G_2}(y_3(k-1),y_1(k))+\alpha_{G_3}(y_2(k-1),y_2(k))$$

$$A_{26}(k)=\alpha_{G_1}(y_1(k-1),y_3(k))+\alpha_{G_2}(y_3(k-1),y_2(k))+\alpha_{G_3}(y_2(k-1),y_1(k)).$$

A pictorial representation of the state transitions given above is provided in Fig. 11. As in the sub-optimal case there are no word-pairs formed at time k=1 so we must define$A_{ji}(1)$ in terms of single word probabilities. Again using the example of state two in a three-input optimal filter, the joint probability is:

$$A_{02}=\log(\Pr[y_1(1)|G_1]\Pr[y_3(1)|G_2]\Pr[y_2(1)|G_3]).$$

Having defined $A_{ji}(k)$ for all i,j, and k, we can proceed with the implementation of the Viterbi algorithm for the optimal filter solution.

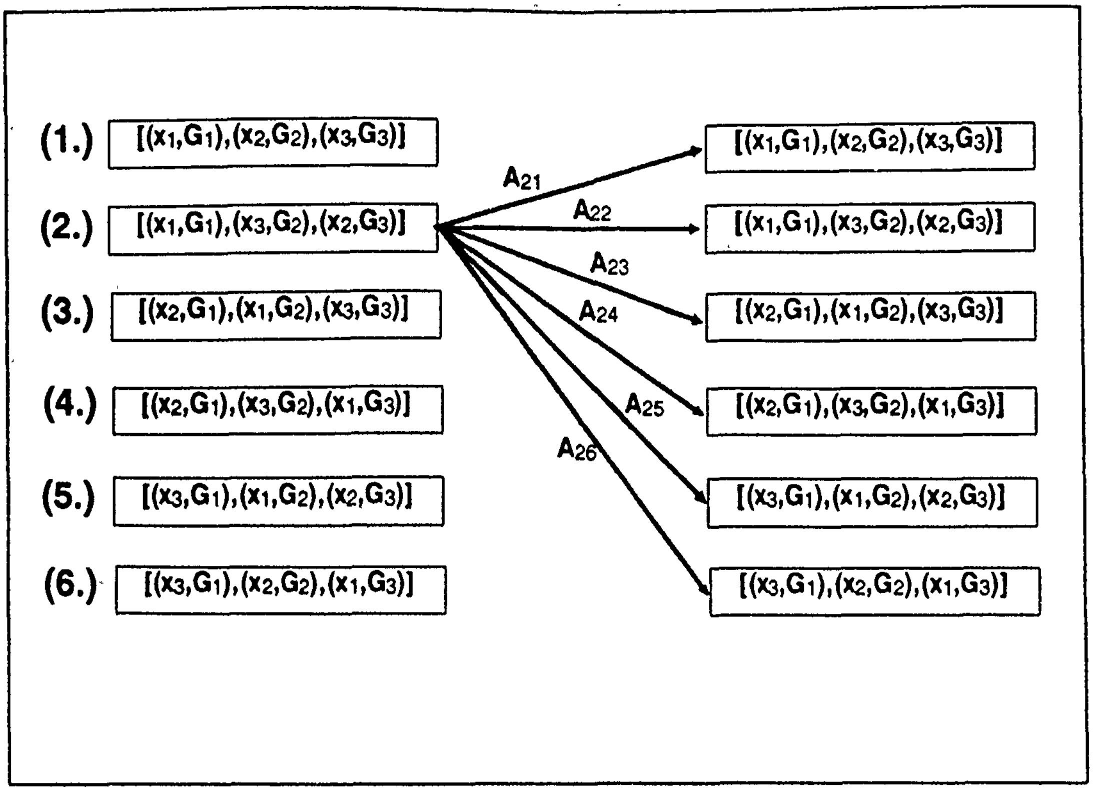
(1.) [(x1,G1),(x2,G2),(x3,G3)]
(2.) [(x1,G1),(x3,G2),(x2,G3)]
(3.) [(x2,G1),(x1,G2),(x3,G3)]
(4.) [(x2,G1),(x3,G2),(x1,G3)]
(5.) [(x3,G1),(x1,G2),(x2,G3)]
(6.) [(x3,G1),(x2,G2),(x1,G3)]
A21
A22
A23
A24
A25
A26
[(x1,G1),(x2,G2),(x3,G3)]
[(x1,G1),(x3,G2),(x2,G3)]
[(x2,G1),(x1,G2),(x3,G3)]
[(x2,G1),(x3,G2),(x1,G3)]
[(x3,G1),(x1,G2),(x2,G3)]
[(x3,G1),(x2,G2),(x1,G3)]

Figure 11 State Transitions for the 3 Input Optimal Filter.

b. The Viterbi Algorithm for the Optimal Case.

The Viterbi algorithm for the optimal speech filter is implemented in exactly the same fashion as for the sub-optimal filter. The algorithm is constructed by substituting the path weight equation (18) for (8), replacing the observation symbol probabilities with the conditional word-pair probabilities like those listed in the previous section, and observing the modifications required to implement the chokepoint refinement. The only modification necessary for the optimal case is in the backtracking step.

When backtracking occurs in the sub-optimal filter all that is necessary to determine which position in the output vector the optimal sequence occupies is to reference the optimal state. A one-to-one mapping occurs between the state and the output position. Since in the optimal filter there are more states than output positions, a many-to-one mapping between state and output position exists. Although the optimal state still uniquely determines the optimal sequence, some decoding must occur before that sequence is known. In order for that decoding to occur, we define G_1 to be the grammar of the desired word sequence and choose the output position of the optimal state sequence which is associated with that grammar. For example in the model of Fig. 10, $y_2(k)$ is the optimal word for the desired sequence if either state 3 or state 4 is selected by the Viterbi algorithm as the optimal state at time k.

c. *Calculation Requirements.*

This section examines the number of calculations required to implement the Viterbi algorithm for the optimal filter case. As in the sub-optimal analysis the calculation requirements are presented in terms of look-ups, floating point operations, and logarithms.

Assume the characteristics of the optimal filter are the same as previously stated for the sub-optimal filter, specifically:

- N input sequences.
- Input sequences have length M.
- Each grammar has a vocabulary of V distinct words.
- Each grammar has a perplexity of Q.

Initialization of the Viterbi algorithm for the optimal filter requires the calculation of joint word probabilities for each of the N! states. Further, since there are N words and N grammars there are N^2 word probabilities which must be established. To determine one word probability requires an average of $\frac{V}{2}$ disk accesses and one floating point operation. The computation of a joint word probability involves multiplication of N of these single word probabilities and computation of the logarithm. This requires another N-1 floating point operations and one logarithm. The resulting average number of calculations required for initialization is therefore $N^2\frac{V}{2}$ look-ups, $(N-1)(N!)+N^2$ flops, and $N!$ logarithms. The recursion equations require $(N!)^2$ joint word-pair probabilities. With N words at time k-1 and N words at time k, N^2 possible word-pairs can be formed. Taking the probability of these word-pairs under the constraints of N grammars gives a total of N^3 word-pair probabilities which must be determined. To find a single word-pair probability requires an average of $\frac{V+Q}{2}$ look-ups and one flop. To arrive at a joint word-pair probability requires that N of these word probabilities be combined and the logarithm taken at the cost of N-1 floating point operations, and one logarithm. The recursion is carried out M-1 times for a total of $(M-1)N^3\frac{V+Q}{2}$ look-ups, $(M-1)(N-1)(N!^2)+N^3$ floating point operations, and $(M-1)(N!)^2$ logarithms. Since all values for $\psi_i(k)$ are stored in memory as they are calculated, there are no floating point operations or look-ups required to perform the backtracking step. Combining all the calculations required in the above steps results in a total of:

$$\text{look-ups} = (M-1)\frac{V+Q}{2}N^3 + \frac{V}{2}N^2 \tag{32}$$

$$\text{floating point ops} = (M-1)(N-1)(N!)^2 + (N-1)(N!) + N^3 + N^2 \tag{33}$$

$$\text{logarithms} = (M-1)(N!)^2 + N! \tag{34}$$

4. Comparison of Optimal and Sub-Optimal Filters.

When comparing the sub-optimal and optimal filters two areas of immediate concern are accuracy and speed. When comparing the accuracy of the two filter types on the theoretical level it is difficult at best to arrive at an analytical equation which will provide a measure of accuracy under general conditions. Any attempt to discover such an equation is beyond the scope of this thesis and accuracy comparisons made here are based strictly upon empirical data. The issue of filter speed is much easier to deal with in a mathematical sense than accuracy. Using the expressions for calculation requirements (29)-(31) and (32)-(34) arrived at earlier in this thesis, a comparison of the theoretical calculation requirements for two-input and three-input filters is presented in Table I.

Table I CALCULATIONS FOR TWO AND THREE INPUT FILTERS.

	Sub-Optimal Filter		Optimal Filter	
Inputs	FLOPS/LOGS	Disk Access	FLOPS/LOGS	Disk Access
2	398/398	101,480	410/398	202,960
3	894/894	227,955	7176/3570	683,865

In Table I the assumptions made are that the grammars used have a vocabulary size, V, of 500 words and a perplexity, Q, of 10.0. The length of the speech sequences to be filtered is 100 words. Assume that each look-up is a disk access which takes on the order of 30ms, an average figure for a hard disk on a PC, and that a floating point calculation takes approximately 500ns, 6 clock cycles on a 12mhz 80286-based PC. If a logarithmic series expansion is used to calculate the logarithm approximately 20 floating point multiplies would be required for a total of 10μs per logarithm [Ref. 7]. Under these conditions the floating point operations and logarithms have little influence on the total time required by the filter since they are several orders of magnitude faster to perform than a disk access. Ignoring the influence of flops and logs, the timing results of Table II may be obtained from the calculation requirements presented in Table I.

Table II EXPECTED TIMING OF TWO AND THREE INPUT FILTERS.

Inputs	Sub-Optimal Filter	Optimal Filter
2	50 min 44 sec	101 min 28 sec
3	113 min 58 sec	341 min 56 sec

From the results presented in Table II it is apparent that in order to achieve anything near real time filtering it is necessary to speed up the look-up procedure by changing storage media and/or establishing a more efficient storage structure.

III. RESULTS AND DISCUSSION

A. Procedures for Gathering of Performance Data.

To test the simultaneous speech separation algorithms developed in this thesis a number of computer programs were written to simulate the algorithms. The source code for these programs, as well as an executable version, written in the ADA programming language, is included on diskette in the Appendix. Table III provides a list of the options available within the simulation program.

Table III SIMULATION PROGRAM OPTIONS.

Option	Function
1	Create the grammar for a text sample.
2	Generate an artificial sentence from a given grammar.
3	Compare two grammars for similarity measure.
4	Filter two inputs using the optimal algorithm.
5	Filter three inputs using the optimal algorithm.
6	Filter two inputs using the sub-optimal algorithm.
7	Filter three inputs using the sub-optimal algorithm.
8	Exit the simulation program.

The first step in the testing procedure was to select sample texts to use in the creation of the grammars. In order to reduce the time required to run the filter programs, the text samples selected for this thesis were limited to small vocabularies, with as much repetition of words and word-pairs within the sample as could be found. Table IV lists the origins of the text samples for which grammars were constructed.

Table IV TEXT SAMPLES USED TO CREATE FILTER GRAMMARS.

Grammar	Source of text.
A.	"Ben Bug", A first grade reading book.
B.	"The Jet", Another first grade reader.
C.	"The Little Engine That Could", A popular childrens book, Original Edition.
D.	"The Little Engine That Could", Retold.
E.	"The Little Engine That Could", Retold again, for younger readers.
F.	Sample from "East of Eden", a novel by John Steinbeck.
G.	Another sample from "East of Eden".
H.	Concatenation of A. and B.
I.	Concatenation of A. through G.
J.	San Jose Mercury News story on Oakland A's baseball game.
K.	San Francisco Chronicle story on San Francisco Giants baseball game.

The text samples were processed, using option 1 of the testing program, to produce the grammars necessary for the filtering operation. Not all these grammars were utilized in the testing of the filters. The two newspaper articles were approximately the same similarity and complexity as the two samples from "East of Eden" and were not processed further.

During the creation of a grammar, a measure of size and perplexity for that grammar is obtained. Once the sperate grammars were created, these grammars were compared against one another to determine the degree of similarity between them. This was done using option 3 of the testing program.

The next step was to use option 2 of the testing program to generate artificial word sequences which had the same statistical characteristics as the sample text. These word sequences are nonsense sentences, but exhibit much of the same grammatic structure as an English language sentence. A typical example of a generated word sequence, in this case based on the grammar of "The Jet", was:

"jet is up. the jet. the jet is mad at bob has a lot of pep.".

This word sequence shows the english sentence structure built in to the statistics of the grammar, but also exhibits the nonsense characteristic that arises when a word is used in more than one grammatical context. In this example sentence the word "bob" is the object of the first half of the sentence but is taken to be the subject of the second half.

The generated word sequences were utilized as input to the filter problem, with the purpose of extracting one of the input sequences through the use of the filter program. Note that the use of generated sentences was necessary in order to provide a large enough set of sample inputs to give the results statistical significance. The original sample text used to create the input grammars was also used as input to the filters, with excellent results. Due to memory limitations in the computer used in this simulation the maximum word sequence length that could be tested was 150 words. Since chokepoints in the algorithm occurred far more frequently than once in 150 words, the word sequences being operated upon were actually much shorter than this limit and the results of the simulation were not affected in any way. The filter problems for this thesis were constrained to either two or three inputs and either the sub-optimal or optimal methodology. These filter variations correspond to options 4 through 6 of the testing program. The results obtained from performing the filtering operations are presented below.

B. Results of the Filtering Operations.

The parameters, accuracy, and speed of the filtering operations performed are given in the following sections. The first step in filter testing is the establishment of grammars for all the inputs to be used in the tests. The parameters of these grammars are required in the analysis of the filter performance results, and so are presented immediately below.

1. Parameters of Grammars used in Speech Separation.

There were nine grammars used in testing the speech separation algorithms. The parameters associated with each of these grammars is presented in Table V.

Table V PARAMETERS OF GRAMMARS USED IN TESTING THE SPEECH SEPARATION ALGORITHMS.

Grammar	Sample Size	Vocabulary Size, (V)	Word-Pairs	Perplexity (Q)
A.	317	55	163	2.96
B.	153	35	86	2.46
C.	1169	251	692	2.76
D.	1357	306	747	2.44
E.	324	123	243	1.98
F.	245	140	233	1.66
G.	396	196	361	1.84
H.	470	63	217	3.44
I.	3961	615	1945	3.16

Not all possible combinations of these grammars were used in testing. Only those combinations of grammars which provided the filters with the widest range of input similarity were used. The similarity measures of those grammar combinations used in the testing process are presented in Tables VI and VII. Table VI shows the similarity measures between the various grammars when used in the context of an optimal filter. Table VII gives the similarity measures for these same grammars, but under the conditions of a sub-optimal filter.

Table VI OPTIMAL SIMILARITY MEASURES BETWEEN FILTER GRAMMARS.

Grammar One	Grammar Two	Unweighted Similarity	Weighted Similarity
A.	B.	14.75%	33.26%
A.	H.	75.12%	88.46%
B.	H.	39.63%	64.27%
A.	I.	8.38%	10.54%
B.	I.	4.42%	6.21%
C.	D.	27.68%	57.68%
C.	E.	14.16%	36.47%
D.	E.	15.52%	40.10%
C.	I.	35.58%	65.64%
D.	I.	38.41%	68.81%
E.	I.	12.49%	35.70%
F.	G.	2.59%	8.41%
H.	I.	11.16%	13.61%

Table VII SUB-OPTIMAL SIMILARITY MEASURES OF FILTER GRAMMARS.

Desired Grammar	Competing Grammar	Unweighted Similarity	Weighted Similarity
A.	B.	37.21%	41.83%
B.	A.	19.63%	29.34%
C.	D.	41.77%	56.52%
D.	C.	45.09%	56.97%
C.	E.	47.74%	54.63%
E.	C.	16.76%	28.83%
D.	E.	54.73%	59.97%
E.	D.	17.80%	3.38%
A.	H.	25.35%	80.85%
B.	H.	16.13%	52.34%
A.	I.	2.83%	12.39%
B.	I.	1.80%	7.90%
C.	I.	12.90%	54.56%
D.	I.	15.73%	57.43%
E.	I.	6.32%	28.30%
H.	A or B.	100%	100%
I.	A,B,C,D, or E.	100%	100%

Notice that grammar I is a super-set of all the other grammars, and that grammar H is a super-set of grammars A and B. These grammars were useful in testing the ability of a filter to discriminate between general conversations and those for which a specific grammar subset may be selected. It should also be noted that the highest degree of similarity, other than for grammars with super/sub-set relationships, was only 27.68% unweighted and 57.68% weighted for the optimal similarity measure, and 54.73% unweighted and 59.97% weighted for sub-optimal similarity measure. These figures were surprisingly low, especially since the three text samples used to create these grammars were three different versions of "The Little Engine that Could", all by the same author.

2. Results of the Speech Separation Tests.

Testing was performed on both optimal and sub-optimal filters using the grammars described above. Analysis was performed to determine the performance of the filters in the areas of speed and accuracy under different grammars. The first algorithm tested was the optimal filter.

a. Performance of the Optimal Filter Algorithm.

The optimal filter was very consistent in its performance for all the grammars tested. Even at the highest levels of similarity the accuracy was better than 99 percent correct for a two input filter. Only when a filter was asked to distinguish between a general speech sequence formed from a super-set grammar and a specific word sequence formed from a subset did performance degrade even slightly. The results of the two-input filter tests for the optimal filter algorithm are given in Tables VIII and IX.

The three-input optimal filter showed accuracy percentages similar to those of the two-input filter. Due to the very slow speed of the three input optimal filter not as many combinations of grammars could be tested, and fewer words were processed in those

Table VIII OPTIMAL FILTER ACCURACY, 2-INPUTS, NON-SUBSET GRAMMARS.

Grammar One	Grammar Two	Words Filtered	Errors	Accuracy
A.	B.	1500	3	99.8%
C.	D.	1500	7	99.5%
C.	E.	1500	4	99.7%
D.	E.	1500	2	99.9%
F.	G.	2100	0	100%

Table IX OPTIMAL FILTER ACCURACY, 2-INPUTS, SUBSET GRAMMARS.

Grammar One	Grammar Two	Words Filtered	Errors	Accuracy
A.	H.	1350	27	98.0%
B.	H.	1350	12	99.1%
A.	I.	1500	0	100%
B.	I.	1500	0	100%
C.	I.	1200	14	98.8%
D.	I.	1350	12	99.1%
E.	I.	1200	4	99.7%
H.	I.	900	1	99.9%

cases that were tested. Both subset and non-subset grammars were applied to the three-input filter with very similar results. The accuracy results of the three-input filter are presented in Table X.

The tables above provide a good representation of just how robust the optimal filter algorithm is under a diverse set of grammars. A graph showing the accuracy of the optimal filter versus the grammar similarities is provided in Fig. 12.

Table X OPTIMAL FILTER ACCURACY, 3-INPUTS.

Grammar One	Grammar Two	Grammar Three	Words Filtered	Errors	Accuracy
C.	D.	E.	900	13	98.5%
A.	B.	H.	900	24	97.3%
C.	D.	I.	750	28	96.3%
A.	E.	I.	1500	3	99.8%

The weighted grammar similarity was utilized in the preparation of the graph shown in Fig. 12. When the results were compiled using the weighted similarity measure, a much clearer appreciation of the relationship between similarity and accuracy was obtained than

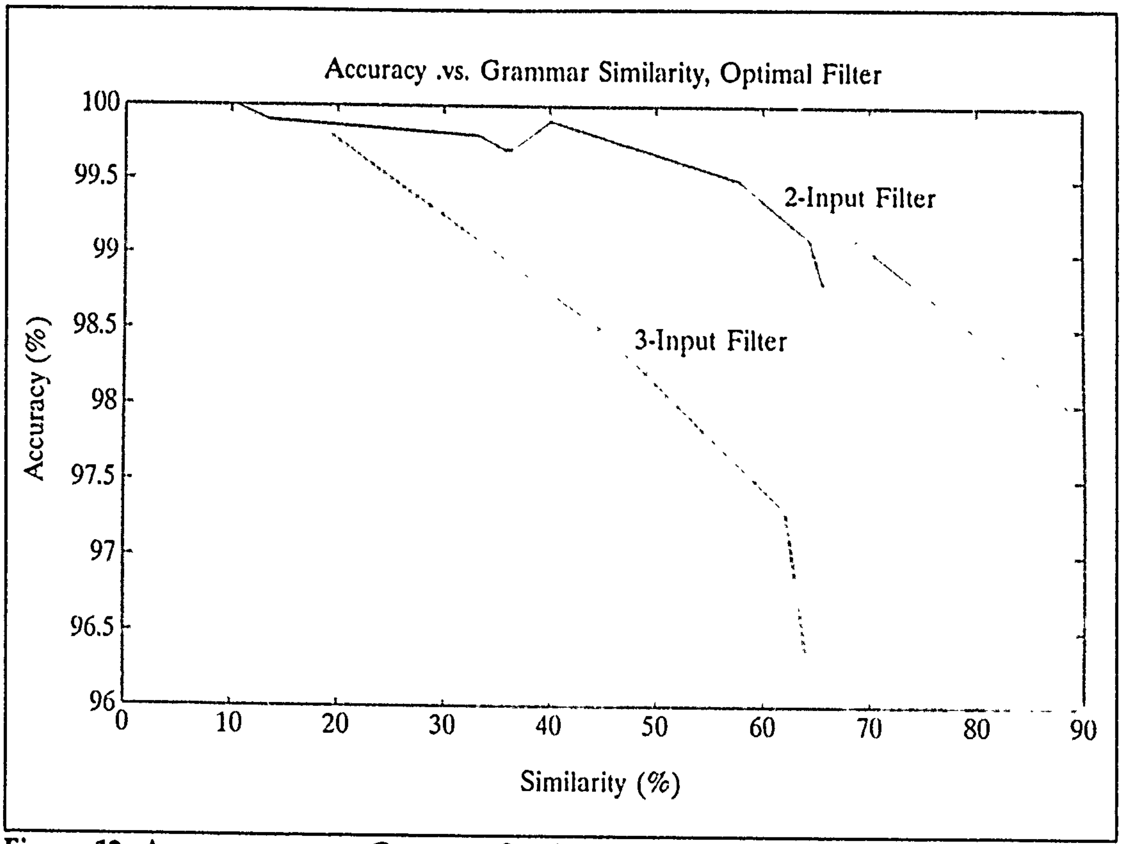

Figure 12 Accuracy .versus. Grammar Similarity, Optimal Filter.

for the unweighted similarity. This graph demonstrates the tendency toward slightly higher error rates as the similarity measure increases, and shows the decrease in accuracy that can be expected from the presence of additional input sequences. Clearly the three-input filter is somewhat less accurate than the two-input filter, but the accuracy of the filter remains quite high. The exceptional accuracy displayed by the optimal filter algorithm is tempered somewhat when the speed of the algorithm is considered.

The greatest drawback of the optimal filter is the speed at which it performs. The large number of look-ups required to obtain all the necessary knowledge from the grammars of the filter slow down the operation greatly. Even at a very modest three inputs this slow down becomes intolerable. The speed of two different grammar pairs was closely observed for both the two and three input filters and a good comparison against theory can be made.

The filter algorithm was run on a 80286 based processor running at 12Mhz. The grammars were stored on hard disk with an average access time of 30 ms. Under these conditions the number of floating point calculations plays very little role in the execution time required by the filter. If it is assumed that a floating point operation takes 500ns, and a logarithm 10μs, there is a 60,000 to 1 ratio between the number of floating point operations and disk accesses that can be performed in a given time period, and a 3000 to 1 ratio of logarithms to disk accesses. For this reason it is safe to ignore the effects of increased numbers of floating point calculations and logarithms when performing the timing analysis of the optimal filter.

The first case to be examined is the two-input filters of grammars A and B, and grammars C and D. A summary of the timing analysis performed on these filter scenarios is given in Table XI. From Table V we see that grammars A and B have

Table XI TIMING ANALYSIS OF THE 2-INPUT OPTIMAL FILTER.

Grammars	Average		Theoretical Time (Sec)	Observed Time (Sec)
	V	Q		
A and B	45	2.71	863	935
C and D	278	2.60	5043	5262
Ratio of C & D to A & B			5.85 : 1	5.63 : 1

vocabularies of 55 and 35 words respectively, or an average of 45 words in each vocabulary. The average perplexity of these grammars is 2.71. Using the formula for the average number of look-ups for an optimal filter, (32), we arrive at a theoretical average of 28,735 disk accesses to filter 150 words. This corresponds to a filter time of 863 seconds, or 5.75 seconds/word. Using the same method to determine disk accesses for the filter using grammars C and D we obtain 168,093 disk accesses required to filter the same number of words using these grammars. This gives a total filter time of 5,043 seconds or 33.61 seconds/word. The ratio of the times calculated for each of these filter scenarios should closely match the ratios of the observed filter times if theory can be held true. The actual times recorded for the filtering of 150 words by each of these filters are 935 seconds for grammars A and B, and 5,262 seconds for grammars C and D. These times are on the same order of magnitude as the theoretical times and the small differences can be accounted for by the time required to execute the program instructions, and the presence of a time consuming video display routine used to monitor the progress of the filter. The theoretical ratio of 5.85:1 is slightly higher than the observed ratio of 5.63:1, but this difference can be attributed to the presence of the common video display routine, which does account for a significant amount of time in each program, but which is not dependant upon the grammars being utilized in the filter.

The extreme increase in time required to filter three inputs using the optimal filter is shown in Table XII. Here an input from grammar H is added to the filter using grammars A and B, and a comparison of the time required to filter the two-input and three-input cases is made. The theoretical time ratio can be arrived at by determining the total number of look-ups for each filter using (32). If we adjust for the difference in look-ups required due to the increased vocabulary and perplexity of the three-input grammar, a time ratio of 3.37:1 is still maintained in favor of the two-input filter. Here again the observed time increase correlates well with theory after taking into account the presence of the video display routine. The large increase in time required for each additional input is most apparent when we look at the time per word required by each filter. The time per word rate of the filter has been increased from 5.75 sec/word to 21.75 sec/word for the three-input filter. This is a steep price to pay for the addition of just one more input. The sub-optimal filter provides some relief from this problem, but at the expense of some degree of accuracy. The results of the sub-optimal filter tests are fully described in the next section.

Table XII TIMING COMPARISON OF THE 2-INPUT AND 3-INPUT OPTIMAL FILTERS.

Grammars	Average		Theoretical Time (Sec).	Observed Time (Sec).
	V	Q		
A & B	45	2.71	863	935
A,B & H	51	2.95	3262	3403
Ratio of 2-Input and 3-Input Times.			3.78:1	3.64:1

b. Performance of the Sub-Optimal Filter Algorithm.

The sub-optimal filter algorithm proved to be quite capable of separating the input speech sequences accurately under most conditions tested. Only when the grammar similarity was above 50% did an appreciable deterioration in filter accuracy occur. These results are presented in Table XIII.

Table XIII SUB-OPTIMAL FILTER ACCURACY, 2-INPUTS.

Desired Grammar	Competing Grammar	Words Filtered	Errors	Accuracy
A.	B.	3000	66	97.8%
B.	A.	3000	45	98.5%
C.	D.	2700	87	96.7%
D.	C.	2700	79	97.1%
C.	E.	2250	105	95.3%
E.	C.	2250	38	98.3%
D.	E.	2250	120	94.7%
E.	D.	2250	12	99.5%
A.	H.	2400	344	85.7%
B.	H.	3000	39	98.7%
C.	I.	1650	66	96.0%
D.	I.	1800	63	96.5%
E.	I.	3000	48	98:4%
H.	A.	1200	652	45.7%
H.	B.	1200	584	51.3%
I.	A.	900	421	53.2%
I.	C.	900	460	48.9%
I.	E.	900	474	47.3%

The biggest difference observed in the accuracies of the optimal and sub-optimal filters was when the sub-optimal filter is asked to extract a word sequence from a grammar which is a super-set of the competing grammar. This presents the sub-optimal filter with a similarity of 100%, and under these circumstances the sub-optimal filter performs no better than tossing a coin or rolling a die.

The three-input sub-optimal filter showed a noticeable decrease in accuracy from that of the two-input filter. This too is a departure from the performance of the optimal filter, but one which should be expected. When inputs are added to the optimal filter there is a corresponding increase in the amount of information, in the form of an additional grammar, which is available to the filter for the decision making process. On the other hand, when we add inputs to the sub-optimal filter we are increasing the number of choices the filter has to decide between, but we provide no additional information to make those choices. The accuracy results of the three-input filter are presented in Table XIV.

Table XIV SUB-OPTIMAL FILTER ACCURACY, 3-INPUTS.

Desired Grammar	Competing Grammar	Competing Grammar	Words Tested	Errors	Accuracy
C.	D.	E.	600	32	94.6%
D.	C.	E.	600	40	93.3%
E.	C.	D.	600	12	98.0%
A.	B.	H.	900	78	91.3%
B.	A.	H.	900	25	97.2%
H.	A.	B.	900	589	34.6%
C.	D.	I.	750	51	93.2%
D.	C.	I.	750	72	90.4%
I.	C.	D.	300	211	29.7%

The accuracy of the sub-optimal filter versus the grammar similarities is presented graphically in Fig. 13. This graph shows the degradation in filter performance as the similarity measure increases, and shows the decrease in accuracy that can be

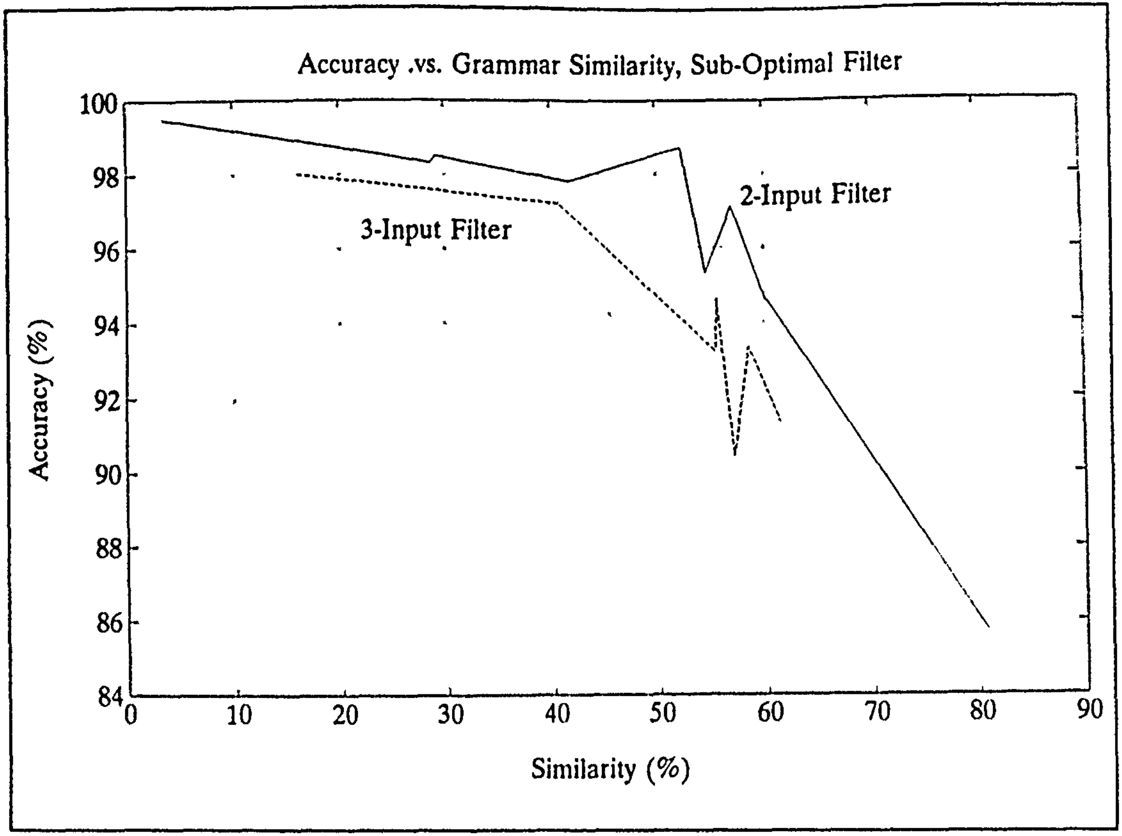

Figure 13 Accuracy versus Grammar Similarity, Sub-Optimal Filter.

expected from the presence of additional input sequences. Notice that at similarities of less than 50% the accuracy of the sub-optimal filter is on par with that of the optimal filter, and when the factor of filter speed is considered the sub-optimal filter becomes very attractive for these filter conditions.

The sub-optimal filter enjoys a significant advantage over the optimal filter in the area of speed. Not only does the sub-optimal filter require far fewer look-ups for any given number of inputs, but there is also much less slow-down in filter operation when the number of inputs is increased. The reason for this becomes apparent from the calculation requirement equations for look-ups, (29) and (32). For the sub-optimal filter algorithm the number of look-ups is proportional to N^2, while for the optimal filter it is proportional to N^3. As for the optimal filter algorithm, the speed of two different grammars were closely observed for both the two and three input filters so that a comparison against theory could be made.

The sub-optimal filter algorithm was run under conditions identical to those for the optimal filter, namely a 80286 based processor running at 12 Mhz, with the grammar stored on a hard disk having an average access time of 30 ms. As before it is safe to ignore the effects of floating point calculations and logarithms when performing the timing analysis. The first cases to be examined are the two-input filters using grammar A, and grammars C. A summary of the timing analysis performed on these filter scenarios is given in Table XV.

Table XV TIMING ANALYSIS OF THE 2-INPUT SUB-OPTIMAL FILTER.

Grammar	V	Q	Theoretical Time (Sec)	Observed Time (Sec)
A	55	2.96	520	594
C	251	2.76	2276	2405
Ratio of C to A			4.37 : 1	4.05 : 1

From Table V we see that grammar A has a vocabulary of 55 words. The perplexity of grammar A is 2.96. Using the formula for the average number of look-ups for a sub-optimal filter, (29), we arrive at a theoretical average of 17,327 disk accesses to filter 150 words. This corresponds to a filter time of 520 seconds, or 3.47 seconds/word. Using the same method to determine disk accesses for the filter using grammar C we obtain 75,872 disk accesses required to filter the same number of words using this grammar. This indicates a total filter time of 2,276 seconds or 15.17 seconds/word. The ratio of the times calculated for each of these filter scenarios should closely match the ratios of the observed filter times if theory can be held true. The actual times recorded for the processing of 150 words by each of these filters are 594 seconds for grammar A, and 2,405 seconds for grammar C. These times are very close to the theoretical times and the differences can be accounted for by the same factors of video display routine and program instruction execution that were experienced in the optimal filter. The theoretical ratio of 4.37:1 is again slightly higher than the observed ratio of 4.05:1, which also follows the pattern observed for the optimal filter. The benefit of the sub-optimal filter is readily apparent when the increase to three inputs is made. The results of increasing the number of inputs to three for the sub-optimal filter using grammar A, is shown in Table XVI.

Table XVI TIMING COMPARISON OF THE 2-INPUT AND 3-INPUT SUB-OPTIMAL FILTER.

Grammars	V	Q	Theoretical Time (Sec).	Observed Time (Sec).
A & B	55	2.96	520	594
A,B & H	55	2.96	1168	1251
Ratio of 2-Input and 3-Input Times.			2.25:1	2.11:1

Here an input from grammar H is added to the filter using grammar A as the desired grammar, and a comparison of the time required to filter the two-input and three-input cases is made. The theoretical time ratio can be deduced by determining the total number of look-ups for each filter using (29). The theoretical time ratios are much smaller than those encountered for the optimal filter, and the observed time increase correlates well with theory after taking into account the presence of the video display routine.

IV. CONCLUSIONS

This thesis served to validate the concept of using grammatic syntax to separate word sequences from different sources, when presented simultaneously to a "listening" filter. The algorithms developed and tested here have displayed high levels of accuracy under the conditions tested.

Using two inputs to the filter, the optimal algorithm was capable of an accuracy greater than 99% for similarity measures up to about 60% where performance began to fall off. Even at higher similarity measures the performance of the optimal algorithm did not fall off rapidly, but slowly lost accuracy. The sub-optimal filter also performed well at low to moderate similarity measures and was comparable to the optimal filter at similarities of less than 50%. The performance of the sub-optimal filter did deteriorate more rapidly at higher similarity values however and the sub-optimal filter failed discriminate between super-set/sub-set grammars.

Although accuracy was high for both the optimal and sub-optimal filter, the sub-optimal filter was shown to be much faster than the optimal filter. Since the accuracy of both filter algorithms is quite high for grammar similarity measures under 50%, the sub-optimal filters increased speed can be effectively utilized if this similarity condition is met. The issue of filter speed is important when attempting to separate simultaneous conversations in real-time. As tested, even the speed of sub-optimal filter does not meet the requirements for real-time filtering.

Under the hardware conditions present in this study it is apparent that neither the optimal nor sub-optimal filter algorithms are capable of operation at rates corresponding to normal conversational speech. If the filter algorithms and storage format of the grammars are left unchanged a tremendous speed-up in hardware would be required to achieve real-time filtering.

A practical general purpose vocabulary for the English language contains approximately 20,000 words and has a perplexity of about 200 [Ref. 3:p. 9]. Using a speech rate of 180 words per minute as the standard by which real time operation is judged, the calculation requirements placed on the hardware system are immense. For the worst-case filter tested in this thesis, the three-input optimal filter, there are 137,700 look-ups, 72 floating point multiplications, and 36 logarithms, that must be performed in the 0.33 seconds allotted for each word filtered. If the processor is the same as described previously the 72 floating point multiplies and 36 logarithms account for 396μs, with the remainder of the time reserved for look-ups. This means that a look-up time of 2.39μs is required for real-time filtering. This is more than a factor of 1000 faster than the current hardware used for look-ups, and is not available using current hard-disk technology. This speed in a look-up might be gained by using RAM to maintain storage of the grammars.

Rather than looking to hardware to improve the speed of the filters, another approach is a modification of the grammar storage method. In this thesis the grammars were stored in ordered linked lists. This data structure has a linear relationship between the size of the grammar and the number of look-ups required to locate a word within that grammar. If a balanced binary tree were to be used to store the grammar this relationship between grammar size and look-ups would be reduced to $\log_2(V)$, where V is the grammar size. For a 20,000 word grammar this means that a word can be located in less than 15 look-ups in a balanced binary tree structure, compared with an average of 10,000 for the linked list structure. This is certainly worth investigating in any future work on this project.

If the issue of a filter speed can be overcome then the next area of investigation in any future research should be in the area of grammars. In this thesis the grammars used for testing were all quite small, both in vocabulary and perplexity. As mentioned above, a practical grammar for general purpose conversations is 20,000 words and a perplexity of 200. The effects of such a vocabulary size and perplexity upon filter accuracy should be fully investigated before the algorithms developed here are considered successful.

In summary, the algorithms developed in this thesis show the potential to play a key role in the development of a system which would allow separation and recognition of multiple simultaneous speech signals. Certainly there is much work to be done at the acoustic signal level before these algorithms can ever be put to use, but once the individual words of the sequences can be recognized the algorithms presented here are quite effective at correctly placing these words with the desired sequence.

APPENDIX

LIST OF REFERENCES

1. James L. McClelland and David E. Rumelhart, "An Interactive Activation Model of Context Effects in Letter Perception: Part 1. An Account of Basic Findings," Psychological Review, Vol 88, No. 5, 1981.

2. James K. Baker, "The DRAGON System - An Overview," Trans. on Acoustics, Speech and Signal Processing, Vol ASSP-23, No. 1, pp. 24-29, 1975.

3. Kai-Fu Lee, Automatic Speech Recognition, Kluwer Academic Publishers, Boston, 1989.

4. A. Bruce Clarke and Ralph L. Disney, Probability and Random Processes: A First Course with Applications, John Wiley & Sons, New-York, 1985.

5. L.R. Rabiner and B.H. Juang, "An Introduction to Hidden Markov Models", ASSP Magazine, Vol. 3, No. 1, pp. 4-16, 1986.

6. Charles W. Therrien, Decision Estimation and Classification: An Introduction to Pattern Recognition and Related Topics, John Wiley & Sons, New York, 1989.

7. Beyer, William H. ed., CRC Standard Mathematical Tables 25th Edition, p. 378, CRC Press, Boca Raton, 1978.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Commandant of the Marine Corps Code TE 06 Headquarters, U.S. Marine Corps Washington, D.C. 20380-0001	1
3.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
4.	Professor Charles W. Therrien Code EC/Ti Electrical and Computer Engineering Department Naval Postgraduate School Monterey, CA 93943	5
5.	Chairman, Code EC Electrical and Computer Engineering Department Naval Postgraduate School Monterey, CA 93943	1
6.	Professor Murali Tummala Code EC/Tu Electrical and Computer Engineering Department Naval Postgraduate School Monterey, CA 93943	1
7.	Capt James L. Kingston USMC Rd 1 Box 52 Tunkhannock, PA 18657	3